Self-Adaptive Software: Landscape and Research Challenges

Prepared by Faisal Sibai for CS 895

Presentation Agenda

- Objective of the paper
- Introduction
- Self-adaptive software: principles and requirements
- Taxonomy of self-adaptation
- Landscape of the research area
- Research challenges
- Summary
Primary Objective

• Survey article which presents a taxonomy based on concerns of adaptation:
  o How
  o What
  o When
  o Where

  Towards providing a unified view of self-adaptation software area.

• The article presents a landscape of research in self-adaptive software.
Introduction

• This paper focuses on self-adaptive software

• Researches have proposed several solutions in this area

• A software application are usually open-loop systems, in self-adaptive they would be converted to a closed-loop system using feedback

• Feedback loop takes into account a more holistic view of what happens inside the application and its environment

• Self-Adaptive systems aim to adjust various artifacts and attributes in response to:
  ○ Self: the whole body of the software in several layers
  ○ Context: everything in the operating environment that affects system properties and behavior

• Feedback occurs from both attributes in the closed-loop system
Introduction

• Why we need self-adaptive software?
  o Increasing cost of handling complexity of software systems to achieve their goals:
    • Complexity management
    • Robustness in handling unexpected
    • Changing priorities
    • Policies governing the goals
    • Changing conditions

• Traditional research concentrates on achieving software development and internal quality. Recently there is a demand to deal with this at operation time (runtime) due to:
  o Heterogeneity level
  o Frequent changes to goals, context...etc. at runtime.
  o Higher security needs.
  o Demand for ubiquitous, pervasive, embedded and mobile applications

• The IEEE-ISO/IEC 14764 2006 standard does not discuss dynamic/runtime changes although it discusses fixing bugs, perfective maintenance...etc.
• Self-adaptive should fulfill requirements at runtime
• Achieved by monitoring the self and the context
Self-Adaptive Software: Principles and Requirements

• **Definition:**
  - **DARPA:** Self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible.
  - **Oreizy:** Self-adaptive software modifies its own behavior in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation.
  - Related point of view regarding the adaptive programming principle as an extension of object-oriented programming
  - Another point of view, adaptation is mapped to evolution, a taxonomy of evolution based on the object of change (where), system properties (what), temporal properties (when), and change support (how)
  - **Lehman:** the essence of self-adaptive software is aligned with the laws of evolution
  - **Kephart:** Self-adaptive software systems are strongly related to other systems like autonomic and self-managing systems. Many use the terms interchangeably
  - **Huebscher and McCann:** similarities and differences between self-adaptive software and autonomic computing
  - **Key point:** life cycle should not be stopped after its development and initial setup. In order to evaluate the system and respond to changes
Self-Adaptive Software: Principles and Requirements

- **Self-* Properties:**

  ![Diagram of self-* properties hierarchy]

  **General Level**
  - **Self-Adaptiveness**

  **Major Level**
  - **Self-Configuring**
  - **Self-Healing**
  - **Self-Optimizing**
  - **Self-Protecting**

  **Primitive Level**
  - **Self-Awareness**
  - **Context-Awareness**

  **Fig. 1. Hierarchy of the self-* properties.**

- **General level:**
  - Global Properties of self-adaptive software (top-down)
  - Subset consists of **self-managing, self-governing, self-maintenance, self-control** and **self-evaluating**
  - Another subset is **self-organizing** emphasizes decentralization and emergent functionalities, usually its bottom-up process.

- **Major level:**
  - IBM autonomic computing initiative defines a set of four properties at this level
  - Have been defined in accordance to biological self-adaptation.
  - Adapt to changes in **context** (changing temperature) or **self** (an injury)
  - Self-configuration, self-healing, self-optimizing, and self protecting.
Self-Adaptive Software: Principles and Requirements

- **Self-* Properties:**
  - Primitive level:
    - Underlying properties: self-awareness, self-monitoring, self-situate, and awareness.
    - Openness and anticipatory are optional but mentioned.
    - **Self-awareness**: the system is aware of its self state and behaviors.
    - **Context-awareness**: the system is aware of its context
  - Relationship with Quality factors:
    - Self-* properties are related to software quality factors
    - Salehi and Tahvildari discuss potential links between the self-* properties and quality factors
    - Self-configuring impacts several quality factors such as:
      - Trainability
      - Functionality
      - Portability
      - Usability
    - Similar discussion for self-optimizing and self-protecting

- **Adaptation Requirements Elicitation:**
  - Six Questions that are important in eliciting adaptation requirements:
    - **Where**: This set of questions are concerned with where the need for change is.
    - **When**: Temporal aspects of change are addressed by this set of questions.
    - **What**: This set of questions identifies what attributes or artifacts of the system can be changed through adaptation actions, and what needs to be changed in each situation
Self-Adaptive Software: Principles and Requirements

• **Adaptation Requirements Elicitation:**
  • **Why:** This set of questions deals with the motivations of building a self-adaptive software application.
  • **Who:** This set of questions addresses the level of automation and human involvement in self-adaptive software.
  • **How:** One of the important requirements for adaptation is to determine how the adaptable artifacts can be changed and which adaptation action(s) can be appropriate to be applied in a given condition.

  o Above questions need to be answered in two phases:
    • **Developing Phase:** deals with developing and building self-adaptive software either from scratch or by reengineering a legacy system.
    • **Operating Phase:** manages the operational concerns to properly respond to changes in the self/context of a software application.
    • Some answered by administrators and managers via polices, others determined by system itself.
    • Distinction between *where* and *what*.

• **Adaptation Loop:**
  o Self-adaptive software embodies a closed-loop mechanisms which is called the adaptation loop.
  o Consist of several processes including **sensors** and **effector**. The loop is called **MAPE-K** loop in the context of autonomic computing.
Self-Adaptive Software: Principles and Requirements

- **Adaptation Loop:**
  
  ![Adaptation Loop Diagram](image)

  - **Adaptation Process:** (operating phase)
    - **Monitoring process:** collecting, partly addresses where, when and what
    - **Detecting process:** analyzing. The when and the where.
    - **Deciding process:** deciding. The what and how
    - **Acting process:** applying. How, what and where to change

  - **Sensors and Effectors:** (operating phase)
    - **Sensors:** monitor software entities to generate a collection of data reflecting the state of the system
    - **Effectors:** rely on in vivo mechanisms to apply changes. effectors realize adaptation actions.
    - Sensors and effectors are essential parts of a self-adaptive software system.
    - Building adaptable software can be accomplished in an engineering or reengineering manner.
**Self-Adaptive Software: Principles and Requirements**

- **Adaptation Loop:**

<table>
<thead>
<tr>
<th>Entity</th>
<th>Technique</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sensors</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logging</td>
<td>Generic Log Adapter (GLA), Log Trace Analyzer (LTA) [IBM 2005]</td>
<td></td>
</tr>
<tr>
<td>Monitoring &amp; events information models</td>
<td>Common Information Model (CIM), Common Base Events (CBE)</td>
<td></td>
</tr>
<tr>
<td>Management protocols and standards</td>
<td>Simple Network Management Protocol (SNMP), Web-Based Enterprise Management (WBEM), Application Response Measurement (ARM), Siena [Carzaniga et al. 2001]</td>
<td></td>
</tr>
<tr>
<td><strong>Profiling</strong></td>
<td>JVM Tool Interface (JVMTI)</td>
<td></td>
</tr>
<tr>
<td>Management frameworks</td>
<td>Java Management eXtension (JMX)</td>
<td></td>
</tr>
<tr>
<td>Aspect-oriented programming</td>
<td>Build to Manage [IBM BtM], Java Runtime Analysis Toolkit [ShiftOne JRat]</td>
<td></td>
</tr>
<tr>
<td>Signal monitoring</td>
<td>Heartbeat and pulse monitoring [Hinchey and Sterritt 2006]</td>
<td></td>
</tr>
<tr>
<td><strong>Design patterns</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Effectors</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Middleware-based effectors</td>
<td>Integrated middleware, Middleware interception [Popovici et al. 2002], Virtual component pattern [Schmidt and Cleeland 1999]</td>
<td></td>
</tr>
<tr>
<td>Metaobject protocol</td>
<td>TRAP/J [Sadjadi et al. 2004]</td>
<td></td>
</tr>
<tr>
<td>Dynamic aspect weaving</td>
<td>JAC [Pawlak et al. 2001], TRAP/J [Sadjadi et al. 2004]</td>
<td></td>
</tr>
<tr>
<td>Function pointers</td>
<td>Callback in CASA [Mukhiya and Glinz 2005]</td>
<td></td>
</tr>
</tbody>
</table>
• **Object to Adapt:**
  o Deals with **where** and **what** aspects of change
    • **Layer:** Which layer of the system can be changed and needs to be changed? Adaptation actions can be applied to different layers.
    • **Artifact and Granularity:** What artifact, attribute, or resource can/needs to be changed for this purpose? Adaptation can change the modules or the architecture and the way they are composed.
    • **Impact and Cost:** The impact describes the scope of aftereffects, while cost refers to the execution **time**, required **resources**, and **complexity** of adaptation actions.
Taxonomy of Self-Adaption

- **Object to Adapt:**
  - **Weak classes:** modifying parameters (parameter adaptation) or performing low-cost/limited-impact actions (bandwidth, load-balancing)
  - **Strong classes:** deals with high-cost/extensive-impact actions. (artifacts)

<table>
<thead>
<tr>
<th>Type</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weak</td>
<td>Caching [Oreizy et al. 1999; Dowling and Cahill 2004]</td>
<td>Caching data, states, connections, objects or components in order to lower the response time, load of servers, or help decentralized management</td>
</tr>
<tr>
<td>Weak</td>
<td>Changing data quality [Mukhija and Glinz 2005]</td>
<td>Changing data quality (i.e., lower resolution) to save bandwidth and increase speed</td>
</tr>
<tr>
<td>Weak</td>
<td>Changing type of data [Cheng et al. 2006; Mukhija and Glinz 2005]</td>
<td>For instance, switching from video to image and even to text to save bandwidth and increase speed</td>
</tr>
<tr>
<td>Weak</td>
<td>Compressing data [Laddaga et al. 2001]</td>
<td>Saving bandwidth by transeiving compressed data</td>
</tr>
<tr>
<td>Weak</td>
<td>Tuning (parameter adjusting) [Karsai et al. 2001]</td>
<td>Adjusting parameters to meet some adaptation goals (i.e., buffer size and delay time)</td>
</tr>
<tr>
<td>Weak</td>
<td>Load balancing [Willebeek-LeMair et al. 1993; Cardellini et al. 1999]</td>
<td>Fair division of load between system elements to achieve maximum utilization, throughput or minimum response time</td>
</tr>
<tr>
<td>Weak</td>
<td>Changing aspects [Pinto et al. 2002; Suvée et al. 2003]</td>
<td>Changing aspect of a component or object with another one with different quality</td>
</tr>
<tr>
<td>Weak</td>
<td>Changing algorithm/method [Oreizy et al. 1999; Robertson and Williams 2006]</td>
<td>Changing the algorithm/ method to meet self-* properties and runtime constraints</td>
</tr>
<tr>
<td>Strong</td>
<td>Replacement, addition &amp; removal [McKinley et al. 2004]</td>
<td>Replacing an entity (e.g., a component) by another one with the same interface but different quality (non-functional)</td>
</tr>
<tr>
<td>Strong</td>
<td>Restructuring/changing architecture [Kramer and Magee 1990; Magee and Kramer 1996; Oreizy et al. 1998]</td>
<td>Changing organization/architecture of the system (it may change the architectural style or design patterns of the system)</td>
</tr>
<tr>
<td>Strong</td>
<td>Resource provisioning [Appleby et al. 2001]</td>
<td>Provisioning additional resources at different levels (this action can be extended to adding/removing any resources, such as servers)</td>
</tr>
<tr>
<td>Strong</td>
<td>Restarting/redeployment [Candea et al. 2006]</td>
<td>Restarting/rebooting (macro- or micro-) or redeployment of system entities at different levels mainly due to faulty/failures</td>
</tr>
</tbody>
</table>
Taxonomy of Self-Adaption

- **Realization Issues:**
  - Deals with how the adaptation can/needs to be applied. Categorized into approach and type classes.

  - **Adaption Approach:** incorporating adaptivity into the system
    - **Static/Dynamic Decision Making:** deals with how the deciding process can be constructed and modified.
      - **Static:** deciding process is hard-coded (e.g., as a decision tree) and its modification requires recompiling and redeploying the system or some of its components.
      - **Dynamic:** decision-making, policies, rules or QoS definitions are externally defined and managed, so that they can be changed during runtime to create a new behavior for both functional and nonfunctional software requirements.
    - **External/Internal Adaptation:** adaption divided into two categories with respect to separation of adaptation mechanism and application logic.

---

**Fig. 4.** Internal and external approaches for building self-adaptive software.
Taxonomy of Self-Adaption

- **Realization Issues:**
  - **External/Internal Adaptation:**
    - **Internal:** intertwine application and the adaptation logic.
      - The whole set of sensors, effectors, and adaptation processes are mixed with the application code, which often leads to poor scalability and maintainability.
      - Can be useful for handling local adaptations (e.g., for exception handling)
      - **Drawback:** costly to test and maintain/evolve, and it is often not scalable
    - **External:** use an external adaptation engine (or manager) containing adaptation processes.
      - Using this approach, the self-adaptive software system consists of an adaptation engine and an adaptable software.
      - Composition of elements in an appropriate architecture and an infrastructure for interoperability are essential
      - **Advantage:** reusability of adaption engine for various applications
      - **Making / Achieving Adaptation:** can be introduced into software systems using two strategies
        1. **Making:** engineer self-adaptivity into the system at the developing phase
        2. **Achieving:** achieve self-adaptivity through adaptive learning

- **Adaption Type:**
  - **Close / Open Adaptation:**
    - **Close-adaptive:** system has only a fixed number of adaptive actions, and no new behaviors and alternatives can be introduced during runtime
    - **Open adaptation:** self-adaptive software can be extended, and consequently, new alternatives can be added, and even new adaptable entities can be introduced to the adaptation mechanism
Taxonomy of Self-Adaption

• **Realization Issues:**
  o **Model-Based / Free Adaptation:**
    • **Model-free:** adaptation, the mechanism does not have a predefined model for the environment and the system itself. Adaptation mechanism adjusts the system. Example: model-free Reinforcement Learning (RL) in adaptation
    • **Model based:** adaptation, the mechanism utilizes a model of the system and its context. This can be realized using different modeling approaches, such as a queuing model for self-optimizing, architectural models for self-healing, or domain-specific models in
  o **Specific / Generic Adaptation:**
    • Some of the existing solutions address only specific domains/applications, such as a database (e.g., IBM SMART project)
    • **Generic** solutions are also available, which can be configured by setting policies, alternatives, and adaptation processes for different domains

• **Temporal Characteristics:**
  • **Reactive / Proactive Adaptation:**
    o **Reactive mode:** the system responds when a change has already happened
    o **Proactive mode:** the system predicts when the change is going to occur. This issue impacts the detecting and the deciding processes.
  • **Continues / Adaptive Monitoring:**
    o Captures whether the monitoring process (and consequently sensing) is continually collecting and processing data
    o **Adaptive** in the sense that it monitors a few selected features, and in the case of finding an anomaly, aims at collecting more data. This decision affects the cost of the monitoring and detection time.
Taxonomy of Self-Adaption

- **Interaction Concerns:**
  - Consists of interacting with humans and/or other elements/systems. The facet is related to all four of the where-when-what-how questions as well as the “who” question

- **Human Involvement (two perspectives):**
  1. The extent to which the mechanism is automated (maturity model).
     - The levels in this model include basic, managed, predictive, adaptive and autonomic.
     - According to this view, human involvement is not desirable, therefore more automation is demanding.
  2. How well it interacts with its users and administrators.
     - Addresses the quality of human interaction to either express their expectations and policies, or to observe what is happening in the system.
     - According to this view, human involvement is essential and quite valuable for improving the manageability and trustworthiness of self-adaptive software

- **Trust:**
  - Trust is a relationship of reliance, based on past experience or transparency of behavior (Security, reliability of humans on adaptive software systems to accomplish tasks)

- **Interoperability Support:**
  - Interoperability is always a concern in distributed complex systems for maintaining data and behavior integrity across all constituent elements and subsystems
  - Global adaptation requirements will be met if elements and designated mechanisms in different layers and platforms of a system are interoperable (e.g., middleware and application).
Landscape of the Research Area

- **Supporting Disciplines:**
  - Shows how different disciplines are able to support and contribute to developing and operating self-adaptive software systems.

- **Software Engineering:**
  - Numerous research areas in software engineering are related to self-adaptive software
  - Ideas developed in the context of software quality for realizing and measuring quality are potentially applicable to self-adaptive software
  - Self-* properties are mostly related to Non-Functional Requirements (NFR), such as security and performance
  - Several researchers have used NFR models, particularly goal models, in self-adaptive software
  - Coupling software with its specification and formal model can allow monitoring correctness and many other metrics with respect to the specified requirements and self-* properties
  - Formal methods provide various ways for modeling software systems as well as utilizing such models.
  - Accordingly, it is possible to rely on formal methods to model adaptable software in adaptation processes.
  - Moreover, formal methods can be used for validation and verification of self-adaptive software to ensure its correct functionality, and to understand its behavior
  - Software Architecture models and languages, such as Architectural Description Languages (ADL), can certainly be helpful in software modeling and management, particularly at run-time
Landscape of the Research Area

- **Supporting Disciplines:**
  - Component-Based Software Engineering (CBSE) can help the development of self-adaptive software in two ways:
    1. It is easier to design and implement an adaptable software relying on component models.
    2. An adaptation engine needs to be modular and reusable, and CBSE can also be used in its development.
  - Another related area, Aspect-Oriented Programming (AOP) and more specifically dynamic AOP, can also be used in realizing self-adaptive software.
    - Facilitates encapsulating adaptation concerns in the form of aspects through dynamic runtime adaptation.
    - Helps in implementing fine-grained adaptation actions at a level lower than components.
  - Service Computing and Service-Oriented Architecture (SOA) can also support realizing self-adaptive software by facilitating the composition of loosely coupled services.
    - Web service technology is often an appropriate option for implementing dynamic adaptable business processes and service-oriented software systems, due to their flexibility for composition, orchestration, and choreography. Propose extensions to the web services architecture to support mission-critical applications.
  - Autonomic Web Processes (AWP), which are web service-based processes that support the self-* properties.

- **Artificial Intelligence:**
  - Not much has been done to apply Artificial Intelligence (AI) techniques, such as planning and probabilistic reasoning, to develop and manage software systems.
  - AI can assist in log/trace analysis and pattern/symptom matching to identify abnormal conditions or the violation of constraints.
• **Supporting Disciplines:**
  
  o **AI** is also rich in planning, reasoning, and learning, which could be useful in the deciding process.
  
  o Obstacle is quality assurance for AI-based systems, which becomes necessary because of the utilized intelligent, heuristic, and search-based techniques.
  
  o Self-adaptive software is based on AI planning: software system plans and may re-plan its actions instead of simply executing specific algorithms.
  
  o Important concept that can be used in self-adaptive software is the way software agents model their domains, goals, and decision-making attributes.
  
  o Important concept: **Multi-Agent Systems (MAS)**, are coordination models and distributed optimization techniques, which can be useful in multi element self-adaptive software.
  
  o Machine Learning and Soft Computing are other areas with the potential to play important roles in self-adaptive software, especially through the “achieving” approach. (analyzing the stream of sensed data and learning the best way to act)
  
  o Genetic algorithms and different on-line learning algorithms, such as Reinforcement Learning (RL) can also be used for this purpose.
  
  o **Decision theory** in both classical and qualitative forms, can contribute to realizing the deciding process.
  
  o **Utility theory**: the quality of being useful” for an action, choice, or alternative, and can be identified either with certainty or with uncertainty (in classical or qualitative form).
  
  o Due to uncertainty, probabilistic reasoning and decision-theoretic planning are required in decision making.
    
    • Markov Decision Process (MDP) and Bayesian network are two well-established techniques for this purpose.
    
    • Applicable to realizing self-* properties due to their uncertain attributes.
Landscape of the Research Area

• **Supporting Disciplines:**
  
  **Control Theory / Engineering:**
  - Similar to self-adaptive software, is concerned with systems that repeatedly interact with their environment through a sense-plan-act loop.
  - The control-based paradigm considers the software system (adaptable software) as a controllable plant with two types of inputs
    - control inputs, which control the plant’s behavior
    - disturbances, which change the plant’s behavior in an unpredictable manner
  - A controller (adaptation engine) changes the values of the plant’s control inputs.
  - The control-based paradigm is often based on a model of the software plant’s behavior

  **Network and Distributed Computing:**
  - Techniques used in network and distributed computing can be extensively applied to self-adaptive software
    - The bulk of the existing software systems are distributed and network-centric
  - Another line of research in this area concerns Peer-to-Peer (P2P) applications and ad hoc networks, which deal with the dynamic change of environment, architecture, and quality requirements.
  - Policy-based management is one of the most successful approaches followed in network and distributed computing
    - Policy-based management specifies how to deal with situations that are likely to occur
  - Most widely used policy type in networks is the action policy (in the form of event-condition-action rules) which is also applicable to self-adaptive software
  - Goal policy (specifying a desired state), and utility policy (expressing the value of each possible state) can also be exploited in self-adaptive software
Landscape of the Research Area

**Supporting Disciplines:**
- **QoS requirements** are related to nonfunctional requirements of a system, and consequently, they can be linked to self-* properties in distributed software systems. Prediction-Enabled Component Technology (PECT)
- **Middleware-based adaptation** would also be applicable to adaptation processes
- Established areas in networks and distributed systems is resource management
- **Monitoring and sensing** techniques have been widely used in networks and distributed systems
  - Basic techniques like heartbeat monitoring and more advanced techniques like pulse monitoring have been used in self-adaptive and self-managing software

**Research Projects:**
- Different academic and industrial sectors to capture main research trends
- Aim to identify the existing research gaps in this area

Please see next slide for the table
Landscape of the Research Area

• **Research Projects:**

  - Sorted based on date of cited publication.
  - Selected on the basis of their impact and novelty/significance of their approach.

<table>
<thead>
<tr>
<th>Projects</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quo [Loyall et al. 1998]</td>
<td>Quality Objects (QuO) provides Quality Description Languages (QDL) for specifying possible QoS states, the system resources and mechanisms for measuring and controlling QoS, and behavior for adapting to changing levels of available QoS at runtime.</td>
</tr>
<tr>
<td>Tivoli Risk Manager [Tuttle et al. 2003]</td>
<td>Providing an integrated security management structure by filtering and correlating the data from different sources and applying dynamic policies.</td>
</tr>
<tr>
<td>Accord [Liu et al. 2004]</td>
<td>Providing a programming framework for defining application context, autonomic elements, rules for the dynamic composition of elements, and an agent infrastructure to support rule enforcement.</td>
</tr>
<tr>
<td>ROC [Candea et al. 2004; Candea et al. 2006]</td>
<td>Building Recursively Recoverable (RR) systems, based on micro-reboot, online verification of recovery mechanisms, isolation and redundancy, and system-wide support for undo.</td>
</tr>
<tr>
<td>K-Component [Dowling and Cahill 2004; Dowling 2004]</td>
<td>A metamodel for realizing a dynamic software architecture based on Adaptation Contract Description Language (ACDL) for specifying reflective programs. ACDL separates the specification of a system's self-adaptive behavior from the system components' behavior.</td>
</tr>
<tr>
<td>CASA [Mukhija and Glinz 2005]</td>
<td>Contract-based Adaptive Software Architecture (CASA) supports both application-level and low level (e.g., middleware) adaption actions through an external adaptation engine.</td>
</tr>
<tr>
<td>DEAS [Lapouchchian et al. 2005]</td>
<td>Proposing a framework for identifying the objectives, analyzing alternative ways of how these objectives can be met, and designing a system that supports all or some of these alternative behaviors using requirements goal models.</td>
</tr>
<tr>
<td>MADAM [Floch et al. 2006]</td>
<td>Facilitating adaptive application development for mobile computing, by representing architecture models at runtime to allow generic middleware components to reason about adaptation.</td>
</tr>
<tr>
<td>M-Ware [Kumar et al. 2007]</td>
<td>Developing middleware to enable agility, resource-awareness, runtime management and openness in distributed applications, by especially addressing performance concerns and business policies.</td>
</tr>
<tr>
<td>ML-IDS [Al-Nashif et al. 2008]</td>
<td>Detecting network attacks by inspecting and analyzing the traffic using several levels of granularity (Multi-Level Intrusion Detection System—ML-IDS), and consequently proactively protect the operating system by employing a fusion decision algorithm.</td>
</tr>
</tbody>
</table>
Landscape of the Research Area

- **Research Projects:**
  - Comparing the previous projects in terms of self-* properties
  - Majority focus on one or two of the known self-* properties
  - Only one project support self-protecting properties (reasons below):
    - Constant changes to network topology
    - Variety of software components/services
    - Variety attacks viruses and increased complexity
    - Most research dealing with self-protecting property focus on the network layer

---

**Table IV.**

Comparing Projects in Terms of Self-* Properties ✓ Supported, - Not Supported

<table>
<thead>
<tr>
<th>Projects</th>
<th>Self-Configuring</th>
<th>Self-Healing</th>
<th>Self-Optimizing</th>
<th>Self-Protecting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quo</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>IBM Oceano</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Rainbow</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Tivoli Risk Manager</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>KX</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Accord</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ROC</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TRAP</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>K-Component</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Self-Adaptive</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CASA</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>J3</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>DEAS</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MADAM</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>M-Ware</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>ML-IDS</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>
## Landscape of the Research Area

### Research Projects:
- How selected projects address adaptation process (No support) to (High support)
- Basis: Efficiency, coverage of different aspects and support for standards
- Vector with four components (Column assessment)

### Table V.
Comparing Projects in terms of Adaptation Processes:
- H (High): Provides explicit features to support the process extensively.
- M (Medium): Provides generic features to partially support the process.
- L (Low): Provides limited support.
- (No Support)—e.g., the vector (2, 5, 7, 2)/16 shows there are 2 “no support,” 5 “low support,” 7 “medium support,” and 2 “high support” projects

<table>
<thead>
<tr>
<th>Projects</th>
<th>Monitoring</th>
<th>Detecting</th>
<th>Deciding</th>
<th>Acting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quo</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>IBM Oceano</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>Rainbow</td>
<td>H</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>Tivoli Risk Manager</td>
<td>–</td>
<td>H</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>KX</td>
<td>H</td>
<td>M</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>Accord</td>
<td>L</td>
<td>L</td>
<td>M</td>
<td>L</td>
</tr>
<tr>
<td>ROC</td>
<td>L</td>
<td>L</td>
<td>M</td>
<td>H</td>
</tr>
<tr>
<td>TRAP</td>
<td>L</td>
<td>L</td>
<td>–</td>
<td>M</td>
</tr>
<tr>
<td>K-Component</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>Self-Adaptive</td>
<td>M</td>
<td>H</td>
<td>H</td>
<td>M</td>
</tr>
<tr>
<td>CASA</td>
<td>M</td>
<td>M</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>J3</td>
<td>M</td>
<td>M</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>DEAS</td>
<td>–</td>
<td>–</td>
<td>M</td>
<td>–</td>
</tr>
<tr>
<td>MADAM</td>
<td>M</td>
<td>M</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>M-Ware</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>L</td>
</tr>
<tr>
<td>ML-IDS</td>
<td>M</td>
<td>H</td>
<td>M</td>
<td>M</td>
</tr>
</tbody>
</table>

### Column-wise Assessment
- (2,5,7,2)/16
- (1,5,7,3)/16
- (2,5,7,2)/16
- (2,3,9,2)/16
## Landscape of the Research Area

- **Research Projects:**
  - Can rely on hybrid approach (Making and Achieving)

<table>
<thead>
<tr>
<th>Taxonomy/Projects</th>
<th>Object to Adapt</th>
<th>Approach</th>
<th>Realization</th>
<th>Type</th>
<th>Temporal</th>
<th>Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quo</td>
<td>application</td>
<td>w m e s c</td>
<td>comp.-based</td>
<td>mb r c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM Oceano</td>
<td>Infrastructure &amp; network</td>
<td>w/s m e d o</td>
<td>data center</td>
<td>mb r c</td>
<td>can be</td>
<td></td>
</tr>
<tr>
<td>Rainbow</td>
<td>application</td>
<td>w/s m e semi-d c</td>
<td>known arch. styles</td>
<td>mb r semi-a</td>
<td>can be</td>
<td></td>
</tr>
<tr>
<td>Tivoli Risk Manager</td>
<td>network &amp; application</td>
<td>system m e - ?</td>
<td>generic</td>
<td>mb r c</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>KX</td>
<td>application</td>
<td>w/s m e d o</td>
<td>generic</td>
<td>mb r semi-a</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Accord</td>
<td>application</td>
<td>w/s m e d c</td>
<td>generic</td>
<td>mb r c</td>
<td>can be</td>
<td>-</td>
</tr>
<tr>
<td>ROC</td>
<td>application &amp; middleware</td>
<td>components/ subsystems w m e s c</td>
<td>comp.-based</td>
<td>mb r c</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TRAP</td>
<td>application</td>
<td>w m e - c</td>
<td>comp.-based</td>
<td>- r c</td>
<td>yes</td>
<td>-</td>
</tr>
<tr>
<td>K-Component</td>
<td>application</td>
<td>w/s m/a e d c</td>
<td>comp.-based</td>
<td>mb r/p c</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Self-Adaptive</td>
<td>application</td>
<td>w/m a e semi-d c</td>
<td>generic</td>
<td>mb r c</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CASA</td>
<td>component</td>
<td>w/m a e semi-d c</td>
<td>comp.-based</td>
<td>mb r c</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>J3</td>
<td>application</td>
<td>w m e s c</td>
<td>J2EE app.</td>
<td>mb r c</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DEAS</td>
<td>application</td>
<td>? m e s c</td>
<td>generic</td>
<td>- - -</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MADAM</td>
<td>middleware</td>
<td>s m e s c</td>
<td>comp.-based</td>
<td>mb r c</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>M-WARE</td>
<td>middleware</td>
<td>parameters w m e s o</td>
<td>generic</td>
<td>mb r c</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ML-IDS</td>
<td>network</td>
<td>parameters w m e d c</td>
<td>generic</td>
<td>mb r/p c</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Research Challenges

- Self-adaptive software creates new opportunities and poses new challenges to development and the operation of software-intensive systems.
- Broad view [Kephart]:
  - Element / Component-level challenges: building element interfaces and contracts to share information.
  - System-level challenges: coordinating self-* properties and adaption process between elements.
  - Human-system interaction challenges: building trust, establishing proper mechanism to involve humans in the adaption loop.
- Provides insight to challenges but does not fit taxonomy discussed earlier.

Fig. 5. Classifying challenges.
Research Challenges

• **Challenges in Engineering self-adaptive Software:**
  o Deals with engineering challenges for requirements analysis, design, implementation and evaluation of self-adaptive software
  
  o **Requirements analysis:**
    • Main task in requirements engineering is to capture the stakeholders' expectations, the key challenges are how to translate, model, and relate those expectations to adaptation requirements and goals to be used at runtime
    • Goal-oriented requirements engineering appears to be a promising approach for addressing these challenges.
  
  o **Design Issues:**
    • How to design self-adaptive software to fulfill the adaptation requirements
    • One needs to extend the existing programming languages, or define new adaptation languages
      o Designing the underlying adaptable software system(s) and adaptation engine(s).
      o forming the architecture connecting them together.
    • How to design adaptable software, either from scratch or from a legacy system?
      o Adaptable software needs to expose important information through sensors (by events), and to facilitate effectors for variability
    • How to use existing experiences in reengineering, reverse engineering, and re-factoriing?
      o Proposing a framework for retrofitting autonomic capabilities into legacy systems. It appears that for variability management, ideas from product line architecture would be useful as well.
    • The issue of interoperability and system-wide architecture.
Research Challenges

**Challenges in Engineering self-adaptive Software:**

- **Implementation languages, tools, and framework:**
  - Building self-adaptive and autonomic software can be accomplished in two ways
    - Extending existing programming languages/systems or defining new adaptation languages
    - Enabling dynamic adaptation by allowing adding, removing, and modifying software entities at runtime
  - There is still a lack of powerful languages, tools, and frameworks that could help realize adaptation processes and instrument sensors/effectors in a systematic manner
  - Middleware-based approach appears more promising, since the infrastructure is available

- **Testing and assurance:**
  - Are probably the least focused phases in engineering self-adaptive software
  - Challenge: The availability of several alternatives for adaptable artifacts and parameters in the system
    - Attempts to use a self-test mechanism at runtime to validate the changes

- **Evaluation and quality of adaptation:**
  - So far there has not been any comprehensive work addressing evaluation criteria or metrics for self-adaptive software, or more generally, for autonomic computing
  - There are links between self-* properties and software quality goals; however, how quality metrics can help measure the quality of adaptation is still an open question
Research Challenges

**Challenges Related to Self-* Properties:**

- **Individual self-* properties:**
  - Self-protecting has received the least attention among the self-* properties.
  - Most of research on self-protecting (not necessarily in the scope of this paper) focus on detecting anomaly symptoms.
  - Some of this research also concentrates on integrating various technologies for security management and recovery.
  - Important question in realizing self-* properties is how well the system is capable of detecting changes and their potential consequences in the adaptable software or its context?
    - Challenges include inferring or predicting the change propagation based on the dynamic model of software.

- **Building multiproperty self-adaptive software:**
  - As shown previously, the majority of projects do not address more than one self-* property.
  - Most of the proposed solutions do not address the relationships between self-* properties including priority, conflict, and the execution order of their actions at run-time.
  - Challenge: coordinating and orchestrating these properties and their derived goals at different levels of granularity.
  - Each self-* property deals with several concerns, such as cost and time.
Research Challenges

• Challenges in Adaption Processes:
  o Monitoring challenges:
    • Challenge: monitoring different attributes in adaptable software is the cost/load of the sensors.
    • A number of in vivo methods collect various information, which may not be required by the desired self-* properties
      o A monitoring process needs to be adapted regarding the adaptable software situation, in order to increase the level of awareness. Such a process can be called an adaptive monitoring process
  o Detecting challenges:
    • Prominent question in the detecting process is “Which behaviors/states of a software system are healthy/normal?”
    • Answering this question often requires a time-consuming static and dynamic analysis of the system, which may also be strongly affected by the underlying random variables (i.e., users' requests arrival times, and faults in different components)
  o Deciding challenges:
    • The deciding process still needs lots of attention both at the local level (adaptation engine) and at the system level.
    • In the presence of multiple objectives, in addition to the necessity of deciding online and dynamically
      o Finding approximately or partially optimal solutions for multi objective decision-making problems
      o Dealing with uncertainty and incompleteness of events/information from the system's self and context
Research Challenges

• **Challenges in Adaption Processes:**
  o Correlating local and global decision-making mechanisms
  o Addressing the scalability and fault-proneness of the decision-making mechanism using centralized or decentralized models.
  
  o **Acting challenges:**
    • Challenge: how to assure that the adaptation is going to be stable and have a predictable impact on the functional and non-functional aspects of the underlying software system
    • Important to know:
      o Whether the adaptation actions follow the contracts and the architectural styles of the system
      o Whether they impact the safety/integrity of the application
      o What will happen if the action fails to complete, or if preemption is required in order to suspend the current action and deal with a higher priority action
    • Formal methods and model-driven solutions, with the aid of model/constraint checking, seem to be a promising direction in this respect.

• **Challenges in Interaction:**
  o At first glance, a human interface for self-adaptive software appears to be much easier to build compared to non-adaptive software. But that’s not the case!
  o **Policy management:**
    • Prominent downside of some of the existing solutions is the lack of explicit representation of policies and goals.
      o Policy translation: policies and goals often need to be decomposed or translated into lower-level/local ones that are understandable by the system elements
Research Challenges

• **Challenges in Interaction:**
  - **Policy management:**
    - *Dynamic policies and goals:* developers need to hard-code or precompile the action selection mechanism for the deciding process.
    - *Rule-based mechanism based on a fixed static conflict resolution mechanism:* commonly used for this purpose.
  - **Building trust:**
    - **Challenge:** how to establish trust.
    - *Not limited to self-adaptive software systems*
    - *Due to its dynamic and automatic nature adds new concerns to the problem*
      - *Independence and intelligence* may make this type of system less traceable for users and stakeholders.
      - *It is essential that a self-adaptive application facilitates trust management for the security concerns,* and also reports its activities and decisions to administrators in order to expose what is going on.
      - *Trust can be built incrementally* to ensure that the adaptation processes are safe and secure.
  - **Interoperability:**
    - *Challenging in most distributed complex systems and particularly in the so-called “systems of systems”*
    - *Coordinating and orchestrating self-adaptation behavior of all elements is a challenging task.*
    - *Fulfilling global requirements and self-* properties, for each property and across different properties,* is *not a straight-forward task*.
Summary

- Self-adaptive software enjoys a growing importance

- In spite of numerous excellent research efforts, this area is still in its infancy, and the existing body of knowledge is far from being adequate to address the escalating demands for self-adaptivity of software in today’s dynamic and ever-changing environments

- Self-adaptive software poses many new opportunities, as well as challenges, for computer scientists and engineers

- This survey article has discussed the basic principles behind self-adaptive software and proposed a taxonomy of adaptation.

- A landscape has been presented based on reviewing a number of disciplines related to self-adaptive software, as well as some selected research projects proposed a taxonomy of adaptation (What, Where…etc.)

- A comparison between the different views of this landscape has provided a framework to identify gaps

- Challenges have been classified into four categories, namely self-* properties, adaptation processes, engineering issues, and interaction