Title	Building Foundations for Dependable Systems (Preliminary Version)
Author(s)	Schlichting, Rick
Citation	
Issue Date	2005-03-11
Type	Presentation
Text version	publisher
URL	http://hdl.handle.net/10119/8277
Rights	
Description	JAIST 21世紀COEシンポジウム2005「検証進化可能電子社会」 = JAIST 21st Century COE Symposium 2005 “Verifiable and Evolvable e-Society”, 開催：2005年3月10日～11日, 開催場所：石川ハイテク交流センター, Technical session 5 <Dependable Infrastructure>

Building Foundations for Dependable Systems

(Preliminary Version)

Rick Schlichting

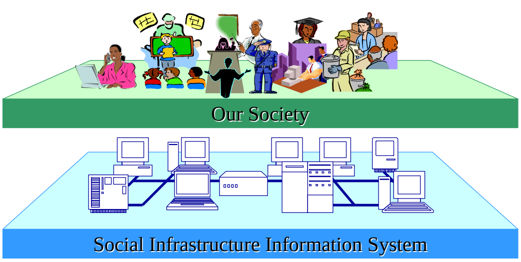
Software Systems Research Department
AT&T Labs-Research
Florham Park, NJ 07932, USA

Work done in collaboration with:

- **Matti Hiltunen (AT&T)**
- **Former Arizona PhD student Jun He (Cisco).**
- **UIUC PhD student Kaustubh Joshi (AT&T VURI intern) and faculty member Bill Sanders.**

Motivation

- **Moving towards an e-society based on information systems and networks.**



Next Generation Information Infrastructure

- **Characteristics**
  - Multiple machines connected by networks.
  - Spectrum of network types and technologies: wired, optical, wireless, ….
  - Spectrum of distances: local-area, metro-area, wide-area,….
  - Spectrum of devices: from sensors to mobile units to high end machines and clusters.
  - Spectrum of applications.
  - Dynamic execution conditions and resource demands.
  - Multiple administrative domains.

➡ **MUST be dependable!**

Dependability

- **Definition: The trustworthiness of a computing system such that reliance to be justifiably placed on the service it delivers.** (Laprie, *et al.*, *Dependability: Basic Concepts and Terminology*, Springer-Verlag, 1992)
- **Includes many properties and attributes.**
  - Reliability
  - Availability
  - Safety
  - Security
  - Timeliness
- **Non-functional or Quality of Service (QoS) attributes.**
  - Focus is not on *how* something gets done, but rather *how well.*
- **Immensely challenging to build software with these attributes!**
  - Failures, intrusions….
  - Concurrent and non-deterministic execution
  - Heterogeneous systems and networks
  - Resource constraints
  - Multiple administrative domains
  - Scale
- **Dealing with multiple attributes makes it even harder *(multidimensional QoS).***
- **Fundamental issue is complexity.**

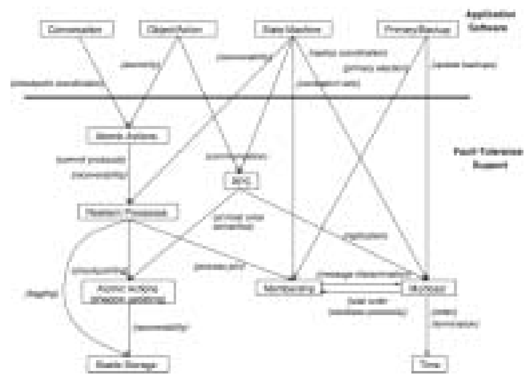
System Abstractions

- ***System abstractions* can simplify the process.**
- **Definition:**
  - Simplified model of a real-life hardware/software component or function.
  - Extracts essential features while omitting unnecessary detail.
- **Goal: Building blocks for constructing more complex systems.**
- **Have long been used to as a way to simplify the design of complex systems.**
- **“Classic” examples:**
  - Process, file, virtual memory,….
  - Layered operating system architectures (e.g., THE system).

➡ **Good abstractions are those that people use without thinking about the underlying implementation.**

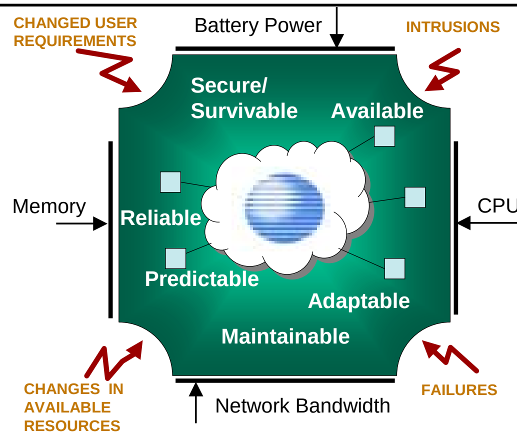
What about Dependability?

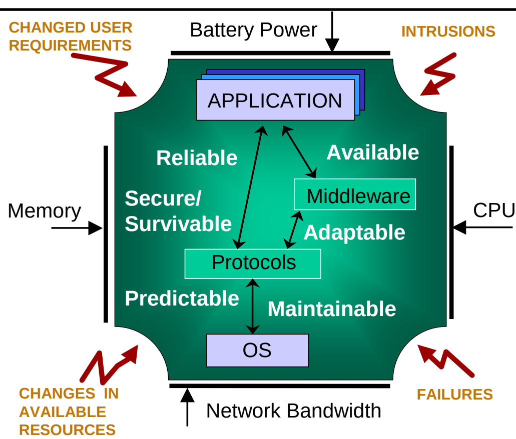
- **Certainly some good dependability-related abstractions.**
  - Provide enhanced QoS characteristics.
- **Hardware virtualization.**
  - Stable storage: abstract storage that never fails.
  - Fail-stop processor: virtual processor whose only failure is a detectable crash.
- **Services for networked systems.**
  - Often focus on providing common global information across machines despite machine and network failures (*virtual shared state*).
  - Implemented as middleware and/or using network protocols.
  - Consistent global clock: abstraction of a single system-wide clock.
  - Atomic multicast: shared message queue
  - Distributed atomic actions (transactions): all or nothing execution across machines.
- **Can also be organized as layers or hierarchies.**



Challenges and Issues

- **Abstraction failures (*leaky abstractions*).**
  - Impossible to implement an abstraction in which QoS properties hold under all conditions.
  - Inherently probabilistic.
- **Composing abstractions.**
  - Reasoning about properties of combinations of abstractions.
  - Conflicts and tradeoffs between different attributes.
  - Performance overhead.
- **Unnecessary attributes.**
  - Matching attributes of abstractions to application and execution environment.
  - Unnecessary attributes can mean extra execution overhead.
- **Changing QoS attributes dynamically.**
  - Providing ability to adapt at runtime





Dependable Systems Research at AT&T

Provide support for building system abstractions and services that bridge the gap between network and application.

- **Support for configurable solutions**
  - Ability to customize properties to the characteristics of the execution environment and the needs of the application.
- **Support for adaptive behavior**
  - Ability to change execution behavior dynamically to react to changes in the execution environment or the application.

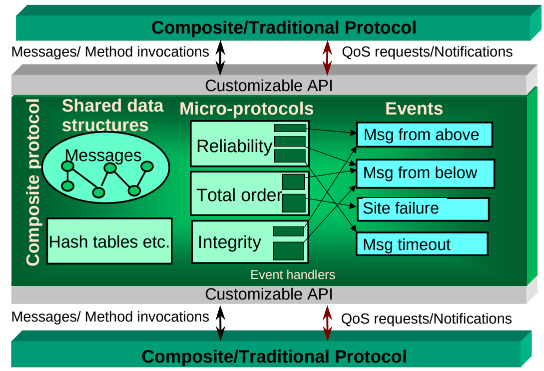
Cactus ⇨ configuration and customization

Cholla ⇨ adaptation

Cactus: Building Highly Configurable Software

- **Both a programming model and an implementation framework for building customized software from collections of software modules.**
- **Highlights:**
  - Fine-grain configuration and customization.
  - Multiple types of attributes and properties, each implemented by a collection of alternative modules.
  - Combination of hierarchical and non-hierarchical composition.
- **Focus:**
  - Communication-oriented services in networks, i.e., protocol stacks and distributed services (but more general).
  - Highly customizable Quality of Service (QoS) attributes related to fault tolerance, timeliness, security, etc. (but useful for other reasons).
- **Addresses challenge of module interaction in highly-configurable software.**

Cactus Approach



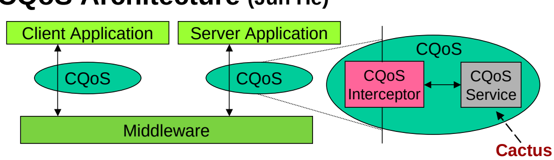
Cactus Model

- **Protocol/service = composite protocol.**
  - Provides service-specific API.
- **Property/QoS attribute = micro-protocol (MP).**
  - MPs interact using an events, shared data, and *dynamic messages*.
  - Mechanisms provide decoupling of MPs ⇒ configurability.
- **Service customization = choose appropriate MPs.**
- **Dynamic adaptation = load/activate/deactivate MPs at runtime.**
- **Two implementations of Cactus 3.0.**
  - C version running on different variants of Unix.
  - Java version.

Example Protocols and Services

- **Configurable Transport Protocol (CTP)**
  - Ordering, reliability, flow/congestion control, security.
- **Secure and Survivable Communication (SecComm)**
  - Privacy, authenticity, integrity, replay prevention, combinations.
- **Configurable Quality of Service (CQoS)**
  - Adding transparent multi-dimensional QoS customization to distributed object systems.
- **Distributed System Monitoring Service (CDSMon)**
  - Function to be monitored.
- **Location-Based Services (LBS)**
  - Functionality based on location for mobile services.
- **Ad-Hoc Networking (AHN)**
  - Dynamic QoS
- **AT&T Enterprise Messaging Network (EMN)**
  - Per request QoS for mobile service platforms
- **Others**
  - RTD channels, group RPC, membership, configurable DSM,….

CQoS Architecture (Jun He)



- **CQoS consists of two components:**
  - Application and platform-specific *CQoS interceptor* generated from IDL.
  - Generic *CQoS service component* implements customizable QoS using Cactus.
- **Micro-protocols include:**
  - Fault tolerance: ActiveRep, PassiveRep, TotalOrder, MajorityVote, Membership, StateRecovery… .
  - Security: DESPrivacy, Authentication, AccessControl …
  - Timeliness: PrioritySched, QueueSched, TimedSched.
- **Semantically different combinations of micro-protocols provide semantically different variations of multi-dimensional QoS.**

Adaptive Systems

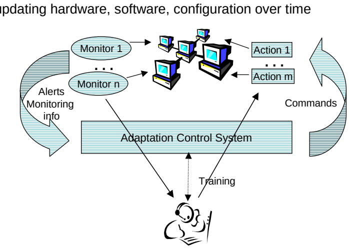
Dynamically changing system behavior.

Motivation:

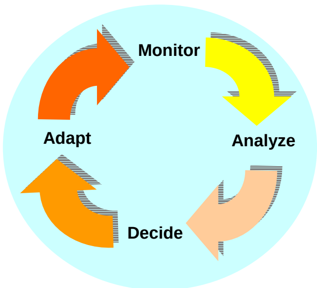
- Short term ⇒ react to changes in the environment: failures, spam/virus/worm attacks, flash crowds, change in wireless connectivity, intrusions
- Long term ⇒ system evolution: updating hardware, software, configuration over time

Adaptive actions:

- Change parameters: router configuration, video frame rate, spam definitions (*value adaptations*)
- Change software modules: video encoder, caching (*algorithmic adaptations*)
- Change resource allocation: bandwidth, CPUs (*resource adaptations*)



Execution Control Feedback Loop



Each phase can be complex in large networked systems:

- Monitoring involves data across multiple hosts and multiple sources.
- Analyzing may involve heuristics or evaluation over time.
- Decision may involve evaluating tradeoffs or distributed algorithms.
- Adaptation may involve distributed coordination across multiple hosts.

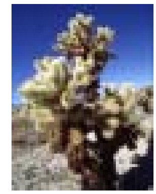
All must be done in a running system and an environment that continues to change.

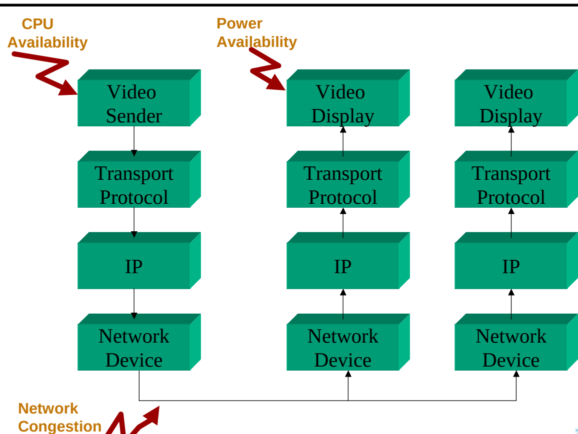
Adaptation mechanisms versus policies:

- Mechanisms provide hooks for monitoring and effecting changes as well as protocols for data collection, analysis, and adaptation coordination.
- Policy encapsulates tradeoff analysis and "business logic".

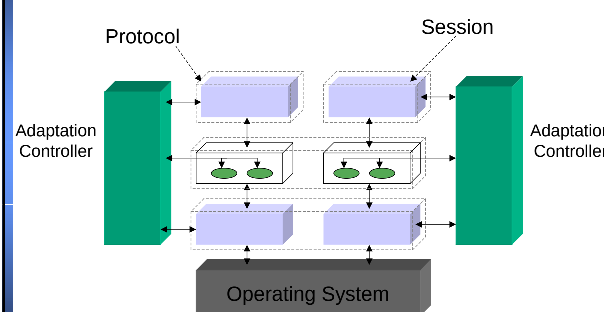
Cholla Adaptation Architecture

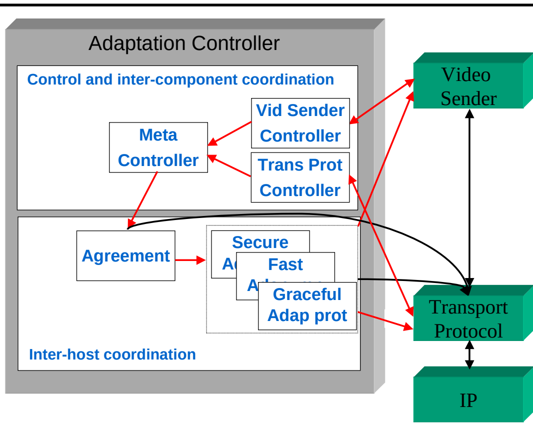
- **Support for value and/or algorithmic adaptations.**
- **Challenges:**
  - Decoupling control from regular functionality.
  - Coordinating adaptations
    - » Inter-component coordination on a single host
    - » Inter-host coordination for distributed services
  - Composition of adaptation policies.
  - Developing appropriate adaptation policies.
  - Efficient realization of policies.
- **Solution: Cholla adaptation architecture**
  - Uses Cactus as underlying platform for implementing adaptive mechanisms and protocols.





Software Architecture



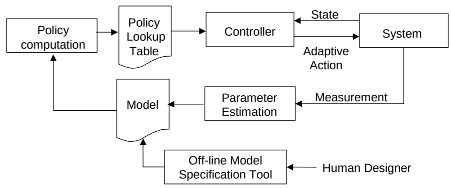


Adaptation Controller

- **Implements execution feedback control loop:**
  - Monitors system state and controls adaptation.
- **Monitoring:**
  - Input variables from controlled components.
  - Input from external monitoring.
- **Control:**
  - Generates outputs based on inputs plus adaptation policies.
  - Changes execution parameters in controlled components (value adaptations).
  - Orchestrates module changeovers (algorithmic adaptations).
- **Implementations:**
  - FLAC: Fuzzy logic based adaptation controller. Focuses on value adaptations and inter-component coordination.
  - CAC: Cactus based adaptation controller. Focuses on algorithmic adaptations and inter-host coordination.
  - Others possible….

Policy Generation
(Kaustubh Joshi, Bill Sanders)

- **Goal: Use stochastic models of system and environment to generate optimal policies for selecting adaptive actions.**



- **Formulation of the problem as a Markov Decision Process**
  - Must deal with state space explosion: state aggregation, model decomposition
- **Currently applying to AT&T EMN system.**

Conclusions and Future Work

- **Useful system abstractions are the key to building a highly dependable information infrastructure for e-society.**
- **Our research is addressing issues related to building such abstractions:**
  - Cactus: flexible fine grain configuration based on two-level composition model.
  - Cholla: Control and coordinated adaptation.
- **Future work**
  - Using Cactus and protocols/services built using Cactus.
  - New protocols for cross-host coordination.
  - Policies, policies, policies!

For More Information

Bhatti, Hiltunen, Schlichting, and Chiu. Coyote: A System for Constructing Fine-Grain Configurable Communication Services. *ACM Trans. on Computer Systems* 16, 4 (Nov. 1998), 321-366.

Chen, Hiltunen, and Schlichting, Constructing Adaptive Software in Distributed Systems. *Proc. 21st Conf. on Distributed Computing Systems (ICDCS),* (April 2001), 635-643.

Wong, Hiltunen, and Schlichting. A Configurable and Extensible Transport Protocol. *Proc. Infocom 2001,* (April 2001), 319-328.

Hiltunen, Schlichting, and Ugarte. Building Survivable Services using Redundancy and Adaptation, *IEEE Trans. on Computers* (February 2003), 181-194.

He, Hiltunen, Rajagopalan, and Schlichting. Providing QoS Customization in Distributed Object Systems, *Software-Practice and Experience* 33,4 (April 2003), 295-320.

He, Hiltunen, Schlichting. Customized Dependability Attributes for Mobile Service Platforms. *Proc. DSN-2004 Dependable Computing and Communication Symp.* (June 2004), 574-583.

Joshi, Hiltunen, Sanders, Schlichting, and Agbaria. Online Model-Based Adaptation for Optimizing Performance and Dependability. *Proc. WOSS '04* (Oct 2004).

Hiltunen, Schlichting. The Lost Art of Abstraction. *Proc. DSN Workshop on Architecting Dependable Systems*, Springer-Verlag LNCS, 2005, to appear.