Automatic Configuration of Component-Based Distributed Systems

Ph.D. Thesis Defense

Advisor: Prof. Roy H. Campbell

May 17, 2000
Introduction

- Modern Society requires software developers to
  - produce large quantities of programs
  - write large, complex programs
  - support different OSes
  - support different machine architectures

- Partial solution:
  - Component Technologies
    - Enterprise Java Beans, ActiveX Controls, CORBA Component Model
Problems in Existing Component Technologies

- Lack support for representing dependencies among components

- Difficult to support
  - Automatic Configuration
  - Dynamic Reconfiguration
  - Fault-tolerance
  - Adaptation, etc...
Lack of Proper Dependence Management in Existing Operating Systems

1. Administration / Configuration
   - Junk libraries left on Windows after uninstall

2. System Architecture
   - Different (static) instances of same OS
   - Configuration of Microkernels

3. Fault-tolerance
   - Module failure not handled by others
Our Solution

- Infrastructure for Dependence Management supporting
  - Automatic Configuration
  - Dynamic Reconfiguration
  - Code Distribution
- Help developers to support
  - Fault-Tolerance
  - Consistent Reconfiguration
  - Adaptation
Presentation Overview

1. Introduction

2. Overall Architecture
   2.1 Automatic Configuration Service
   2.2 Component Configurators
   2.3 Reconfiguration Agents

3. Applications

4. Experimental Results

5. Related Work

6. Future Directions and Conclusions
Architecture

Manages two kinds of dependencies:

1. **Prerequisites** - requirements for loading a component into the system runtime.

Overall Architecture

- Prerequisite Specifications
- Automatic Configuration Service
- Component Configurators
- Mobile Reconfiguration Agents
- CORBA services
- QoS-Aware Resource Management
1. Automatic Configuration Service

1. Fetches component code and prerequisites from a *Component Repository*.

2. Dynamically link component code into the application address-space.

3. Based on the prerequisites, repeats the process for other components.
Prerequisites

- What a component needs to run:
  - nature of hardware resources
  - share of the hardware resources
  - software services (i.e., components) it requires

- Video Client example:
  - PC with Sound card
  - 50% of 300MHz CPU
  - software component with MPEG decoder
  - CORBA Video Service
Automatic Configuration Architectural Framework

- load application
- return reference

Prerequisite Resolver

- fetch prerequisites
- fetch components

Component Repository

Prerequisite Parser

QoS-Aware Resource Manager

Cache
2. Component Configurators

- Reify dynamic inter-component dependencies.
- Created on-the-fly by the Prerequisite Resolver.
- System and application software can inspect and reconfigure the Dependence Graph.
Component Configurator Framework

- Allows browsing, inspection, and reconfiguration
- Can be customized through inheritance
- Clear separation of concerns
ComponentConfigurer

Implementation

- Single-process applications: Java and C++
- Distributed applications: CORBA

interface ComponentConfigurer {
    void addHook (in string hookName);
    void deleteHook (in string hookName);
    void hook (in string hookName, in ComponentConfigurer cc);
    void unhook (in string hookName);
    void registerClient (in ComponentConfigurer client, in string hookNameInClient);
    void unregisterClient (in ComponentConfigurer client, in string hookNameInClient);
    void eventFromHookedComponent (in ComponentConfigurer hookedComponent, in Event e, in unsigned short timeToLive);
    void eventFromClient (in ComponentConfigurer client, in Event e, in unsigned short timeToLive);
}
Customizing Component Configurators

- **Synchronization**: LockingConfigurator
- **Attributes**: ComponentConfiguratorAttrib

- Application-specific customization to support:
  - fault-tolerance
  - consistent reconfiguration
  - adaptation
3. Reconfiguration and Inspection with Mobile Agents

- Suitable for Large-Scale Systems

- Agents may carry
  - graph
  - reconfiguration script
  - state
  - results
Presentation Overview

1 Introduction
2 Overall Architecture
   2.1 Automatic Configuration Service
   2.2 Component Configurators
   2.3 Reconfiguration Agents
3 Applications
4 Experimental Results
5 Related Work
6 Future Directions and Conclusions
Applications of the Architecture

- dynamicTAO
- Multimedia Distribution System
- Developed by other researchers:
  - LegORB
  - 2KQ and QoS-aware VoD service
  - SIDAM: road traffic information system
  - CORBA Persistent Object Service
  - Distributed Chess Game
  - Gaia OS for Active Spaces
  - 2KFS
Application: \textit{dynamic TAO}

- CORBA-compliant Reflective ORB
- Extension of TAO (Washington University)
- Uses Component Configurators to support
  - inspection
  - reconfiguration
- Interaction with the reflective interface can be done
  - using a point-to-point connection
  - using mobile agents
dynamic TAO Structure

Servant1Configurator -> Servant2Configurator

TAOConfigurator ->
  - ConcurrencyStrategy
  - SchedulingStrategy
  - SecurityStrategy
  - MonitoringStrategy

DomainConfigurator
Application: Scalable Multimedia Distribution

- **Goal**: stream multimedia to millions of users over the Internet.

- The system can be used with
  - Live Multimedia Streaming
  - Stored Content Streaming
  - Audio/Videoconference

- **Approach**: use a wide-area network of *Reflectors*
A Reflector Network
Applying the Architecture

- Prerequisites and AutoConfig Service
  - Used to customize the components of each Reflector
  - Reserving memory, CPU, bandwidth (not implemented)

- Component Configurators
  - Represent intra- and inter-Reflector dependencies
  - Support fault-tolerance
Presentation Overview

1 Introduction
2 Overall Architecture
   2.1 Automatic Configuration Service
   2.2 Component Configurators
   2.3 Reconfiguration Agents
3 Applications
4 Experimental Results
5 Related Work
6 Future Directions and Conclusions
Experimental Results

- Experiments with the three elements of the architecture
- Testbed:
  - 2 Sun Sparc Ultra-60, two 360MHz CPUs
  - 5 Sun Sparc Ultra-5, 333MHz CPU
  - Solaris 7 OS
  - 100Mbps Fast Ethernet
  - third experiment: Internet
1. AutoConfig Service
Loading Several Components
AutoConfig Service
Loading Components of Different Sizes
AutoConfig Service

Loading Components of Different Sizes

Time (ms)

Component Size (Kbytes)

- loading code
- saving code
- fetching code
- saving prereq
- fetching prereq
- autoconf protocol
2. Dynamic Reconfiguration Using Component Configurators
Impact of Dynamic Reconfiguration on QoS
3. Mobile Agents for Reconfiguration and Inspection

- **Testbed:**
  - Three Sparc Ultras, Solaris 7 @cs.uiuc.edu
  - Three 333MHz PCs, Linux RH6.1 @escet.urjc.es
  - Three 300MHz PCs, Linux RH6.1 @ic.unicamp.br

- 100Mbps Fast Ethernet (intra-domain)
- Public Internet (inter-domain)
Mobile Agents vs. Conventional Client/Server

Round-Trip Time (s)

Number of Commands

- agents
- client/server
Uploading a New Component to 9 Nodes
Conclusion of the Experiments

- The three elements of our architecture can be implemented efficiently.
- Can improve the performance of existing systems.
Related Work

- Prerequisites:
  - Job Control Languages [IBM 65]
  - SOS operating system [Shapiro 94]
  - QoS description languages [Frølund 99]

- Automatic Configuration:
  - Customizable Operating Systems
  - Jini
Related Work

- Component Configurators
  - Reflection
  - Software Architectures (ADLs)

- Dynamic Reconfiguration based on
  - Software Buses [Hofmeister 93]
  - Connectors [Taylor 98]
  - Workflow applications [Wheater 98]
Original Contributions


4. *dynamic TAO* [IFIP/ACM Middleware’2000]

5. Multimedia Distribution System [ICAST’98]
Future Work

- Libraries of Component Configurators
- Dynamic Adaptability
- Integration with ADLs
- Security
- Reconfiguration as atomic transactions
- Automating Prerequisite generation and verification
Summary

- This thesis has
  1. presented an architectural framework for dependence management in component-based distributed systems,
  2. described a concrete implementation of the architecture,
  3. presented two applications that utilize the architecture, and
  4. described experiments and analyzed the performance of the implementation.
As computing devices become pervasive in our society, we will encounter highly dynamic environments, complex dependencies, and potentially difficult management. This thesis presented an integrated architecture that addresses these problems in a clean and efficient way.
int WebBrowserConfigurator::eventOnHookedComponent
    (ComponentConfigurator *cc, Event e)
{
    if (cc == JVMConfigurator)
    {
        if (e == REPLACED)
            try {
                FrozenObjs fo = currentJVM->freezeAllObjs ();
                currentJVM = JVMConfigurator->implementation ();
                currentJVM->meltObjects (fo);
            }
            catch (Exception exp)
                throw new ReconfigurationFailed(exp);
        
        else ...
    }
**dynamic TAO Architecture**
Distribution Tree vs. Point-to-Point
Overall Architecture
(relationships)
Simple Prerequisite Description Format (SPDF)

Hardware requirements:
- **machine_type**: SPARC
- **os_name**: Solaris
- **os_version**: 2.7
- **min_ram**: 5MB
- **optimal_ram**: 40MB
- **cpu_speed**: > 300MHz
- **cpu_share**: 10%

Software requirements:
- **filesystem**: CR: /sys/storage/DFS1.0 (optional)
- **networking**: CR: /sys/networking/bsd-sockets
- **windowmanager**: CR: /sys/WinManagers/simpleWin
- **jvm**: CR: /interp/Java/jvm1.2 (optional)
Application-Specific Customization: Web Browser Example

A WEB Browser Component Configurator and its relation to system Component Configurators
2. Dynamic Reconfiguration Using Component Configurators

- Events triggering reconfiguration:
  1. Reflector shutdown, kill, or \texttt{ctrl-c}
  2. Errors leading to Seg. Fault or Bus Error
  3. Reconfiguration message sent by sysadmin
  4. Sudden machine crash or network disconnection

- 1, 2, and 3 can use the dependency info

- Our experiments use the \texttt{ctrl-c} option