Internetware
A Software Paradigm for Internet Computing

Gang Huang
Key Laboratory of High Confidence Software Technologies, Ministry of Education
Institute of Software, Peking University
hg@pku.edu.cn
Agenda

• Internet As A Computer and its Distinguished Software Characteristics

• Challenges Addressed by Internetware

• Internetware Research and Practice
Internet as a Computer (Internet Computer)

- Internet is evolving to a Global Ubiquitous Computer
  - Many big and hot trends in IT research and business try to study such evolution from different perspectives

**Technical Trend**
- Semantic Web
- Social Computing
- Service Computing
- System of Systems
- Pervasive Computing
- Grid/Cloud Computing
- Internet of Things

**Big Trend**

**Business Trend**
- Digital Economy
- E-government
- Internet Culture
- Social Network
- Modern Service
- Virtual World
- Smarter Planet

• Grid/Cloud computing proposes a new model of networked applications from the perspective of resource sharing and management.
• Pervasive computing discusses a new situation of networked applications from the perspective of human computer interaction.
• Service Oriented Computing focuses on a new form of software with emphasis on collaboration and dynamism from the philosophy of software as a service.
• …
Scenarios of Internet Computer

Cooperation On Demand (emergent)

happens anytime anywhere among anyone (just like the Internet).

involves partners with no or little relations before and after.

Global Monitoring for Environment & Security

C2 centres

Survey tower

Sensors

Civil Security

Unmanned aerial vehicle

Endangered civilians

Fire-fighting planes

Modelling centres

Sensors - UAV – C2 – D & M Centres Workflow

Data centres

(maps...)

Teams on the field

Buoys

(sensors)
Such emergent cooperations enabled by software are ad hoc, labor-based and untrusted

- Before “Internet as a Computer”, the goal of cooperation is predefined or predicted while the partners and interactions can be fixed or not
- In the era of “Internet as a Computer”, more and more goals are unclear, unexpected or undeterministic before the cooperation finally happens (e.g. real-world emergency)
“Internet Computer” brings many distinguished characteristics to software systems for implementing new business naturally with new technology.

**Cooperative**: software can interact with others in static, dynamic and even on demand manners.

**Situational**: software is capable of perceiving its runtime context and scenarios.

**Autonomous**: software is relatively independent of others; it can perform operations as it will and adapt itself when necessary.

**Evolvable**: software is easy to add, remove and change its functionalities on-the-fly and just-in-time.

**Emergent**: software may have un-designed behaviors or un-expected effects on its runtime instances or interactions with others.

**Trustworthy**: software should promise some kind of tradeoff among process quality, internal system quality, external system quality and usage quality.
Existing software paradigms* cannot well support those new characteristics of software on Internet computer, and then Internetware comes to being.

* Software model and its construction theory from the perspective of software engineers or programmers.
Challenges to Internetware

Programming Paradigm (what to be)
• abstracts the elements and their relationships of a software system
• Internetware model should
• leverage legacy software and new characteristics
• Enable open collaboration between components
• Adapt itself for emergent contexts and situations

Engineering approach (how to make)
• Systematically control the software development, deployment, maintenance and evolution
• Internetware engineering should
  • Identify the self-organized communities and domains or facilitate the self-organizations
  • Satisfy requirements via collaborating existing and/or emergent components
  • Involve all stakeholders, especially the actual end users

Programming Language & System (how to be)
• incarnates the elements and their relationships of a software model
• Internetware middleware should
• Provide a container for instantiating and operating Internetware components
  Provide collaboration mechanisms.
• Equip legacy software systems with Internetware characteristics
• Enable context-awareness and reflection

Quality assurance (how to be good enough)
• Focal points of software quality change from system-centric to usage-centric
• Internetware quality assurance should define quantitative and qualitative evaluation framework for quality
• Assure the quality via engineering approach at development time as well as middleware at runtime
Internetware Research in China

IT-evolved real-world

Software Model (what to be)

Infrastructure (how to be)

Engineering Approach (how to do)

Peking University (PI)

Tsinghua University

Institute of Software of CAS

East China Normal University

Nanjing University

Academy of Mathematics and Systems Science of CAS

Researchers from IBM, SJTU, DLTU, etc.

• “Theory and Methodology of Agent-based Middleware on Internet Platform”
  
  – The first national basic research program (973) project on software
  
  – From 2002~2008; > 80 faculty; 17 post-doc; 88 Ph.D; 364 Master

• “High Confidence of Internetware”
  
  – IBM joined as the first foreign company in 973 program
  
  – From 2009~2013; > 100 faculty
Internetware R&D Outputs

Internetware Engineering Approach

Internetware Middleware Framework

Internetware Software Model

Internetware Platform and tools

Internetware Demo and Case Study

Bottom-up Knowledge-Driven engineering

On demand Construction/cooperation

Architecture centric

Agent-like Entity

Self-organizing Domain/community

Middleware for Runtime

Internetware Characteristics

Cooperative, Situational, Emergent, Evolvable, Autonomous, Trustworthy
For every emerging paradigm, we usually solves WHAT-IS and HOW-TO at first (in 5-10 years) and then HOW-WELL.
Roadmap of High Confidence Internetware

**Quality Aware Internetware Engineering Approach**

- Self-Adaptive and Self-Organizing Internetware Middleware (via Models at Runtime)
- Internetware Quality Model and Evaluation Approach

- Extend existing outputs with focus on killer applications of Internetware and High Confidence in emerging computing and application paradigms, like cloud computing, mobile Internet, internet of things, cyber-physical systems, social networking, etc.
Models@Runtime are models causally connected to the state and behavior of a runtime system.

At first constructed from monitoring the state and behavior of (is-be) a system at runtime for quality of service, then going to control the state and behavior of (to-be) models at runtime.

At first constructed from designing the state and behavior of (to-be) models at design time for functionality and quality, then going to update the state and behavior of (is-be) a system at runtime.
Reference Architecture of M@RT

Models @ Runtime

M3: models for interoperation, integration and management

M2: definition of models, relations and constraints (context, analysis, plan, configuration, performance, reliability, etc.)

M1: constraints on models & relations (consistency, timeliness, etc.)

- derived models
- models (read-only)
- models (read-write)

Causally connected

defined by

defined by

derived from

System @ Runtime

M0: system

M3: meta meta model; M2: meta model; M1: model at runtime; M0: system
SM@RT: Supporting Models at Runtime

- **SM@RT**: A model-driven framework for constructing the causal connection between the architectural models and runtime systems in an automated manner (using MOF/QVT standards)

  - **models at runtime**
    - for quality of service
    - at first constructed from monitoring the state and behavior of (is-be)
  - **derived runtime architectures**
    - (satisfy management needs)
    - causally connected via model synchronization
    - inherently runtime architecture (conform to design architecture)
    - causally connected via automatically generated reflective code

  - **a system at runtime**
    - then going to control the state and behavior of (to-be)
Construction of Inherent Runtime Architecture

1) define MOF-based meta model of inherent runtime architecture (conform to design architecture)

2) select the manageability of a system at runtime

3) define QVT-based mapping between the meta model and system manageability

causally connected via automatically generated reflective code
3) define QVT-based mapping between the meta model and system manageability

mapping SysElement::getX(prop:Property): result:Object
when {self.type=Any and prop=Any}
{ var aux:=self.parent.auxiliary;
  Management mgmt = $aux.getMainEntry();
  $result = mgmt.getAttribute($self.core, $prop.name); \} }

mapping SysElement::addX(prop:Property, v:Object): result:Object
when {self.type=MBeanServer; prop=ejb}
{ var fileName=v.fileName;
  var server:=self.parent.auxiliary.server;
  String[] signature={"java.lang.String"};
  String[] params=new String{$fileName};
  String deployedName=(String)mgmt.invoke($server,
    "deployJar", params, signature);
  $result = ObjectName.newInstance(deployedName); \} }

What operation of meta model
Under what meta model's condition
for which model elements
Actual invocation to the manageability

when get/set the value of a model element, if necessary, invoke the corresponding management APIs
Construction of Inherent Runtime Architecture

4) generate the code based on Eclipse M2M and JET

```java
public String getJndiName() {
    try{
        MEJB mainEntry=(MEJB)
        PkuasmanagementPackageImpl
            .getMainEntry();
        ObjectName core=getCore();
        Object res=mainEntry
            .getAttribute(
                getCore(),"jndiName");
        return (String)res;
    } catch(Exception e) {
        return null;
    }

    public Integer getMaxInstancePool(){
        try{
            MEJB mainEntry=(MEJB)
            PkuasmanagementPackageImpl
                .getMainEntry();
            ObjectName core=getCore();
            Object res=mainEntry
                .getAttribute(
                    getCore(),"maxInstancePool");
            return (Integer)res;
        }
    }
```

JET template for the getX methods

```
<%@ start %>
...
public <%=genFeature.getImportedType()%>
 <%=genFeature.getGetAccessor()%>{
...
    try{
        String coreType=genClass.getEcoreModelElement()
            .getEAnnotation("http://sei.pku.edu.cn/core_type")
            .getDetails().get("name").toString();

        String entryType=genPackage.getEcoreModelElement()
            .getEAnnotation("http://sei.pku.edu.cn/entry_type")
            .getDetails().get("name").toString();

        <%=entryType%> mainEntry=(<%=entryType%>)
        <%=genPackage.getPackageClassName()%>.getMainEntry();

        <%=coreType%> core=getCore();

        //TODO: insert platform-specific accessing logic here
        Object res=mainEntry.getAttribute(
            core,"<%=genFeature.getSafeName()%>”);
        return (<%=genFeature.getImportedType()%>)res;
    }
    catch(Exception e){
        e.printStackTrace();
        return null;
    }
    ...
<%@ end %>
```
Construction of Inherent Runtime Architecture

1) Define MOF-based meta model of inherent runtime architecture

2) Select the manageability of a system at runtime

3) Define QVT-based mapping between the meta model and system manageability

4) Generate reflective code

---

```
package javax.management;

Provides the core JMX classes.
See: Description

Interface Summary
Descriptor This interface
DescriptorAccess This interface
DynamicMBean Defines the interface
ManagementContext Can be implemented after b
Manageable

1) define MOF
2) select the manageability of
3) define QVT
4) generate reflective code
```

---

```
public String get2ndName() {
    try {
        MBean mainEntry = MBean
        .getExtension().getMainEntry();
        ObjectName coreGetCore();
        ObjectName extMainEntry
        .getAttribute()
        .getCore()."get2ndName"();
        return (String)xxx;
    } catch (Exception e) {
        return xxx;
    }
}

public Integer getMainInstancePool() {
    try {
        MBean mainEntry = MBean
        .getExtension().getMainEntry();
        ObjectName coreGetCore();
        ObjectName extMainEntry
        .getAttribute()
        .getCore()."getMainInstancePool"();
        return (Integer)xxx;
    } ...
```
Case Studies of Inherent Runtime Architecture

JEE (JonAS/PKUAS, Apusic) Inherent RSA MM
305ele+28map+310loc=22151loc

Eclipse SWT
19ele+23map+178loc=11209loc

PLASTIC
6ele+13map+547loc=9126loc

Android
87ele+95map+431loc=21732loc

IoT Devices
29ele+15map+267loc=8732loc
Construction of Derived Runtime Architecture

if the JEE administrators prefer C2-style architecture for runtime evolution, then they have to derive C2 RSA from the inherent one.

derived runtime architectures (satisfy management needs)

causally connected via model synchronization

inherent runtime architecture (conform to design architecture)

causally connected via automatically generated reflective code

inherent RSA

a system at runtime
Construction of Derived Runtime Architecture

1) define MOF-based meta model of acquired runtime architectures (satisfy management needs)
2) select the target meta model of
3) define QVT-based mapping between the two meta models

causally connected via model synchronization

inherent runtime architecture (conform to design architecture)

top relation Component2DataSource {
    name: String;
    maxPool: Integer;
    enforce domain arc arch: Architecture();
    enforce domain arc conn:Connector{
        parent = arch, name = 'jdbc';
    }
    enforce domain arc comp:Component{
        below = conn, name = name, maxPool = maxPool;
    }
    enforce domain sys server: MBeanServer();
    enforce domain sys data: JDBCDataSource{
        name = name, parent = server,
        jdbcMaxConnectionPool = maxPool;
    }
    when {Root2Root (arch, server);}
}
Construction of Derived Runtime Architecture

4) synchronize two models via incremental bi-directional model transformation

3) Backward transformation to feed the sync result back to the view

1) Forward transformation to calculate the meaning of view changes

4) Recheck if all the original view modifications have been propagated

2) CVS-like three-way comparison to filter out conflicts
Case Studies of Derived Runtime Architecture

Acquired C2 RSA on JEE
29ele+157map

Garlan's C/S RSA on JEE
17ele+73map

SM@RT Lab on IoT
36ele+20map
QVT-based RSA Manipulation Language

detect 38 anti-patterns in 3 JEE systems: 8/ECperf, 7/Petstore, 5/Rubis

calculate component's impact on system reliability using SBRA

```java
transformation Locate_Key_Comp(in jonas:JOnAS) main(){
  var components := jonas.rootObjects[Server].selectOne(true).map toComponents;
  var scenarios := components.map toScenarios;
  var cdg := new SBRA().Create_CDG(components,scenarios);
  var compNames:=components.collect(name);
  var key=""; var maxReliability:= new SBRA().Algo();
  compNames->forEach(curr){
    var cdg_cp := cdg.clone();
    var comp:=cdg_cp.components->SelectOne(name=curr);
    comp.reliability:=0.5 + comp.reliability/2; //increasing the reliability of a given component
    var reliability:=new SBRA().Algo(cdg_cp); //re-calculate the system reliability
    if(reliability>maxReliability){maxReliability:=reliability; key:=curr; }
  }
  return curr; }
```
Future Work: SM@RT Cloud (as killer app)

If system management is the core of business, it may be a killer app of SM@RT.
If online evolution is the core of business, it may be a killer app of SM@RT.
Thank You!