

Reasoning about the Security of Open Architecture Software Systems

Walt Scacchi and Thomas Alspaugh
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3455 USA
14 December 2012

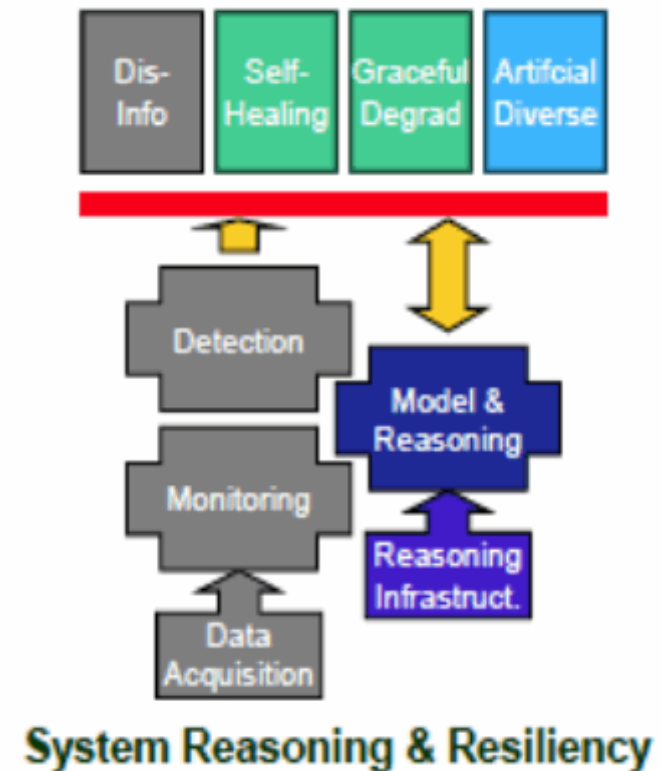
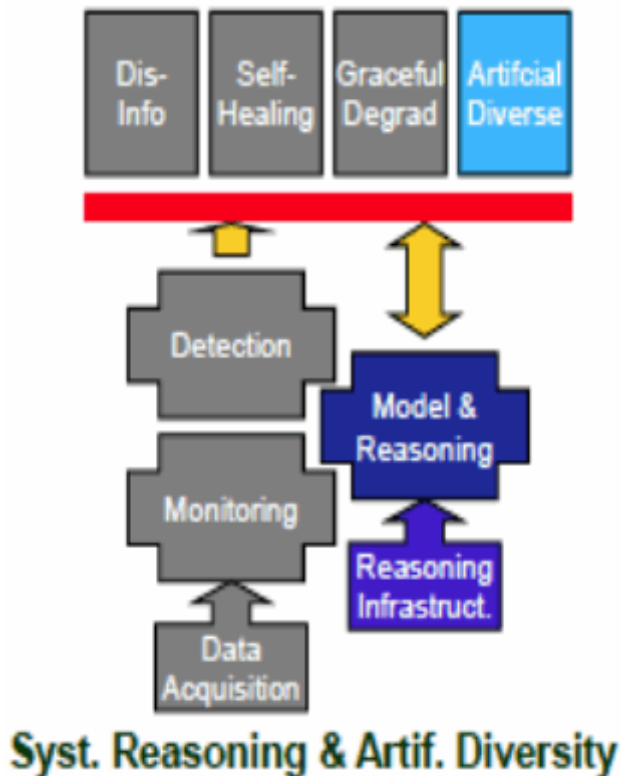
Overview

- Cyber security challenges of interest
- Reasoning about secure Open Architecture (OA) systems
- Examples of recent or work-in-progress results

Cyber security challenges of interest

- Automated reasoning about OA system security
 - Explained in following slides
- Secure Web and Web Applications
 - Shown in demonstration case study later
- Security assuring software development environment
 - Shown in demonstration case study later

Our goal is to support reasoning about OA system security, artificial diversity, self-healing, and graceful degradation



Cyber security challenges of interest

- Automated reasoning about OA system security via robust and resilient autonomic host services
 - System self-awareness
 - Self-diagnosis (static/dynamic monitoring)
 - Self-healing (dynamic OA reconfiguration)
 - Supporting graceful degradation and artificial diversity via
 - *Functionally equivalent variants* of OA system elements
 - *Functionally similar versions* of OA system elements

System self-awareness

- Security of OA systems can be formally specified in affordable, scalable manner
 - Human readable, computer processable description of obligations and rights for access control, encryption, system updates, policies, etc.
- Requires a systematic, transparent *language* (notation) for specifying (modeling) OA system security
 - Further requires meta-model based ontology for OA system security constructs, vocabulary, logic, processing scripts, etc.
- Language model and meta-model represent relationships among elements stored in a knowledge-base (repository)
 - Relations define schemata, object attribute values stored in repository.
- We have prior research in object/knowledge-based modeling, analyzing, and simulating of self-aware software system processes

System self-awareness--*language*

- Language describes *meta-data* about elements within the system's architecture (structure) and security constraints (annotations):
 - OA system elements:
 - Components (open source software or executable binaries)
 - Connectors (data protocols, databases, middleware)
 - (Sub)System configurations (software only, s/w+h/w+net platform)
 - Ecosystem (developers, users, integrators, attackers)
 - Who, what, where, when, why, how, and with what meta-data attributes and values for each system element.

System self-awareness--*language*

- Language describes *meta-data* about elements within the system's architecture annotated with security constraints (structural model):
 - Who (actor), what (action), where (location), when (time), why (belief/policy), how (process), with what (mechanism), to what (element)
- Reasoning entails multiple modes of model analysis
 - *Static analysis*: consistency, completeness, traceability, correctness of model
 - *Dynamic analysis*: mining/filtering of computational results from monitored system (interface) execution model/states
 - *Evolutionary analysis*: detection of intended (planned) and unintended (suspicious) changes to system models, elements, or states
 - *Query processing*: retrieval of results or deductive inference from rules constraining updates to model elements, or to system execution states of interest

Sample of our prior efforts in developing languages for reasoning about features of software system architectures

Choi, S. and Scacchi, W. (2003). Formal Analysis of the Structural Correctness of Software Life Cycle Descriptions, *Intern. J. Computers & Applications*, 25(2), 91-97.

Narayanaswamy, K. and Scacchi, W. (1987). A Database Foundation for Supporting the Evolution of Large Software Systems, *J. Systems and Software*, 7(1), 37-49.

Alspaugh, T.A., Scacchi, W., and Asuncion, H.A. (2010). Software Licenses in Context: The Challenge of Heterogeneously Licensed Systems, *J. Association for Information Systems*, 11(11), 730-755.

Scacchi, W. and Alspaugh, T.A. (2012). Understanding the Role of Licenses and Evolution in Open Architecture Software Ecosystems, *Journal of Systems and Software*, 85(7), 1479-1494.

Scacchi, W. and Alspaugh, T.A. (2013). Advances in the Acquisition of Secure Systems Based on Open Architectures, *Cyber Security and Information Systems J.* 1(2), (to appear).

Also see our References section.

Self-diagnosis (static/dynamic monitoring)

- Static/dynamic analysis of OA system security obligations and rights in a new architectural language.
 - Annotated system has a “security architecture”
 - This architecture serves as a (design-time) *reference model*
 - Reference model guides dynamic run-time analysis of system
 - Run-time analysis determines matches, mis-matches, and conflicts with security constraints in reference model
 - Diagnostic classification rules and triggered processing scripts specify actions to take (logging, reporting, hot swap system run-time configuration, shift execution to different processor cores, etc.)

Self-healing (dynamic OA reconfiguration)

- It is now possible to develop a new language to specify, design, build, package, and deploy concurrent, multi-version systems using:
 - Multiple *functionally equivalent* executable variants
 - new patented compilation technology developed at UCI
 - Multiple *functionally similar* component versions
 - Alternative releases of software component
 - Alternative producers of competing components
 - Ex. Web browsers: Internet Explorer, Google Chrome, Apple Safari, and Mozilla Firefox
 - Ex. Office apps: MS Word, AbiWord, Google Docs, OpenOffice Write
 - Concurrent software build processes enable creation of OA system configured with different variants or versions for one or more processors (or processor cores)

- Self-healing (dynamic OA reconfiguration)
- Availability of multiple concurrent system variants or versions can enable hot/cold swapping of executable run-time systems
 - Requires automated processing scripts and configuration rules
 - Such scripts and rules need to be part of the OA system modeling language
 - Multiple alternative configurations can be stored in repositories in ready-to-swap state, or ready to be adapted within remote target system.

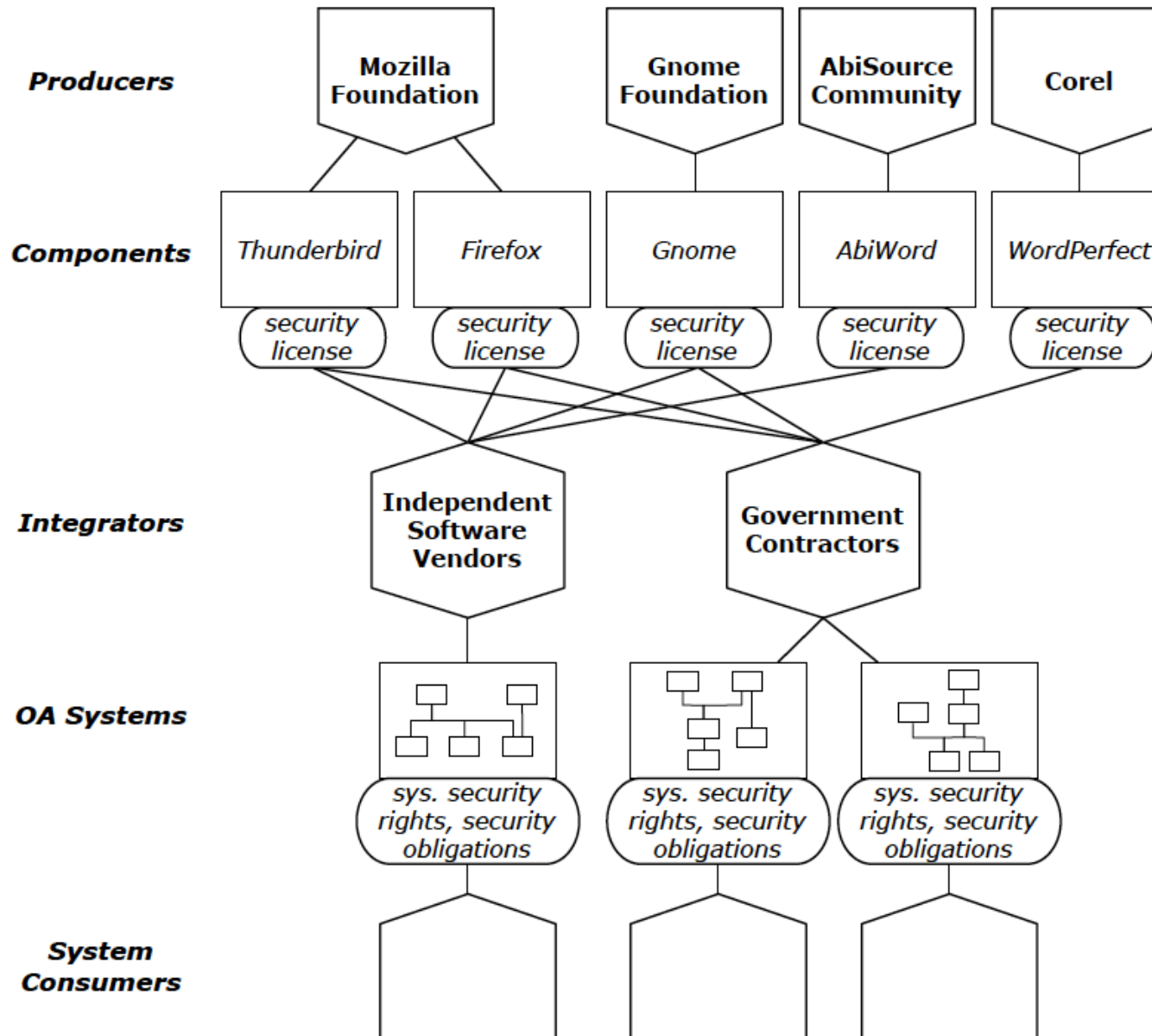
Supporting graceful degradation and artificial diversity

- Hot (dynamic) or cold (static) swappable OA elements
- *Functionally equivalent variants* of OA system elements
 - e.g., executable components with different memory maps, created via multiple concurrent compilations
- *Functionally similar versions* of OA system elements
 - e.g, software product line component instances from different producers
 - e.g., multiple concurrent system configurations with different interconnection layouts

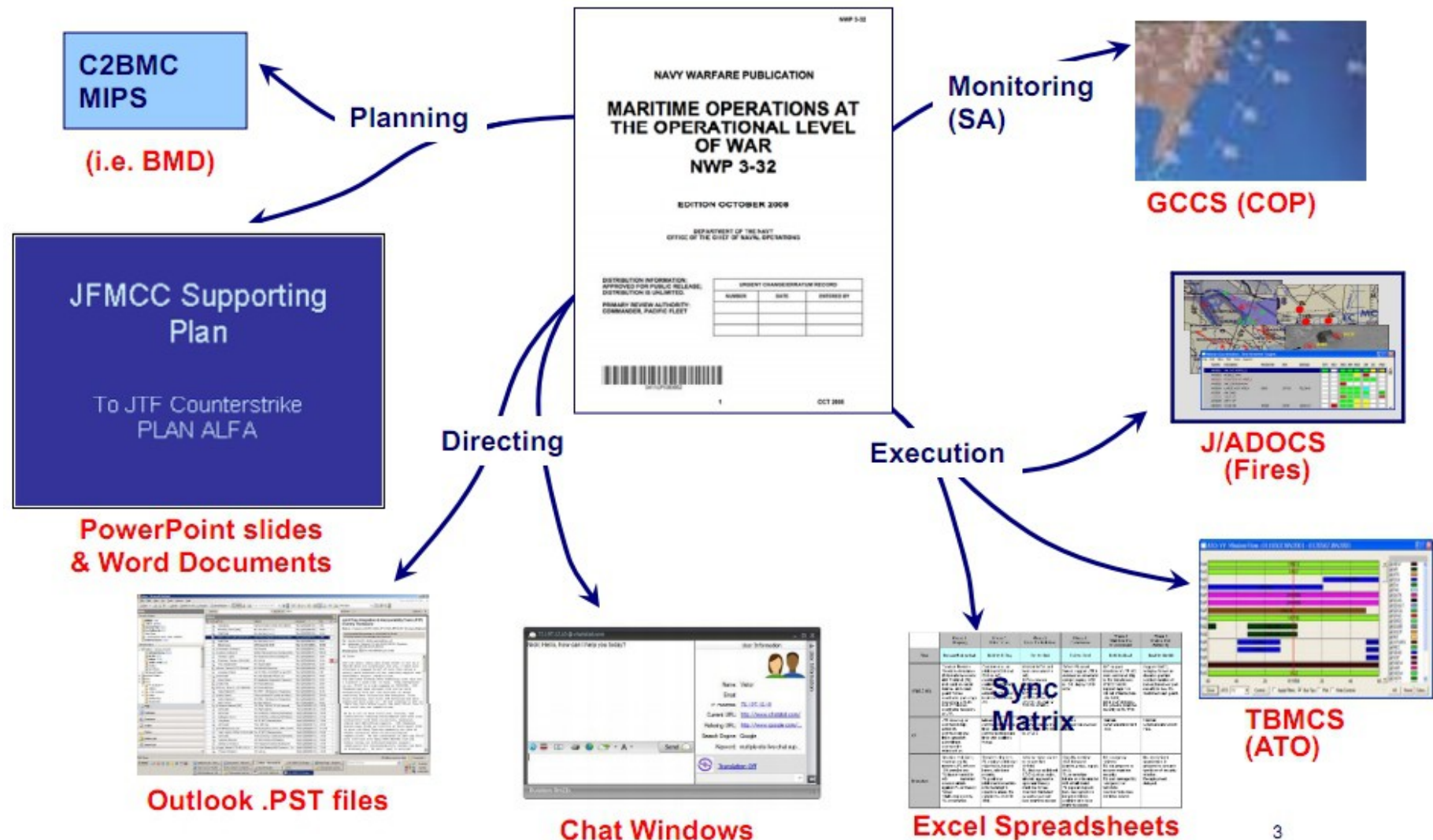
Examples of recent or work-in-progress results

- Language for modeling and meta-modeling as basis for reasoning about OA obligations and rights (simple constraints—not security)
- Tools and techniques for automated reasoning about OA obligations and rights integrity
- OA system software development environment
 - Based on *Eclipse* with *UCI ArchStudio* and analysis plug-in modules

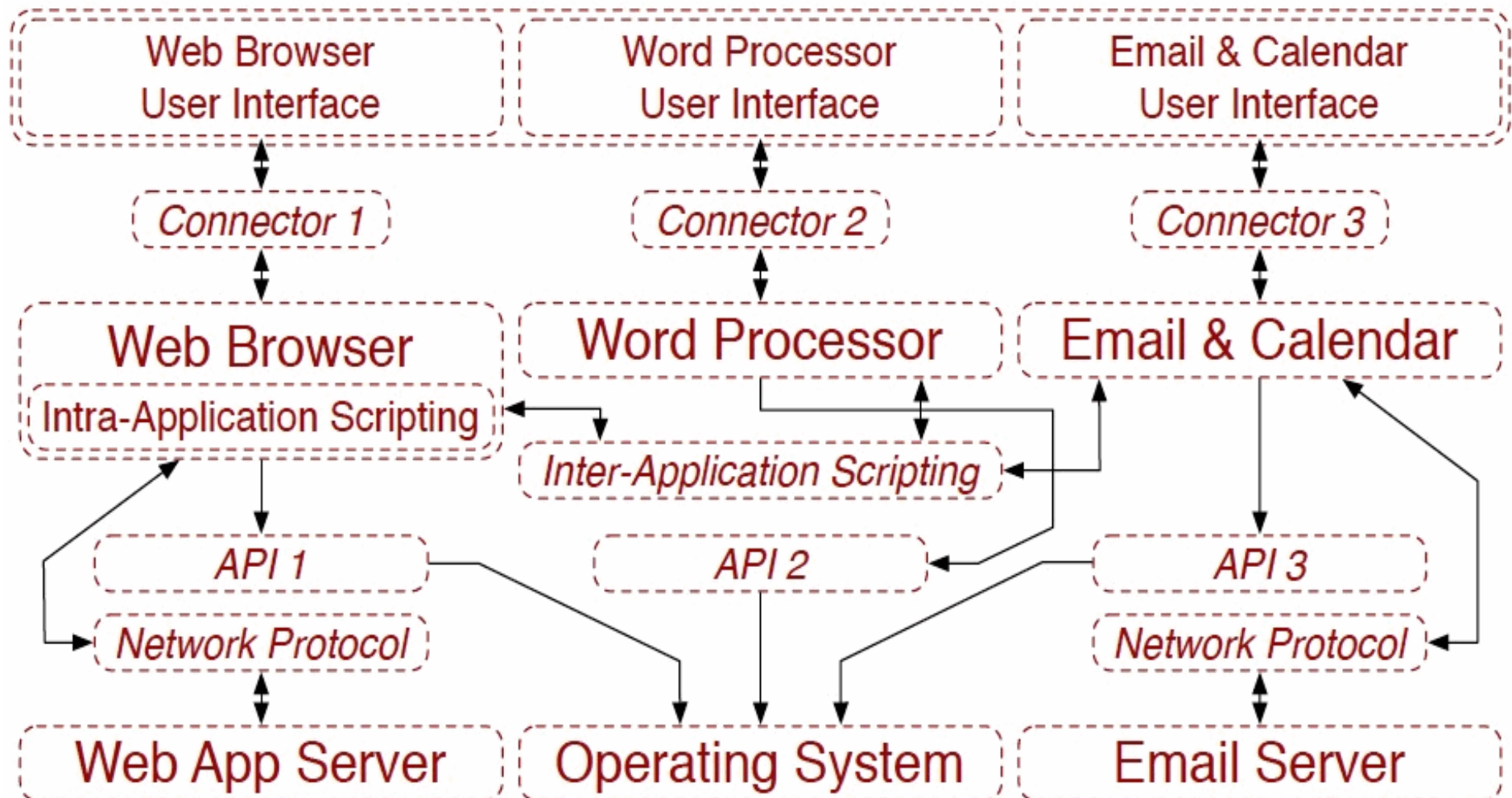
Software ecosystem of OA system producers, integrators, consumers



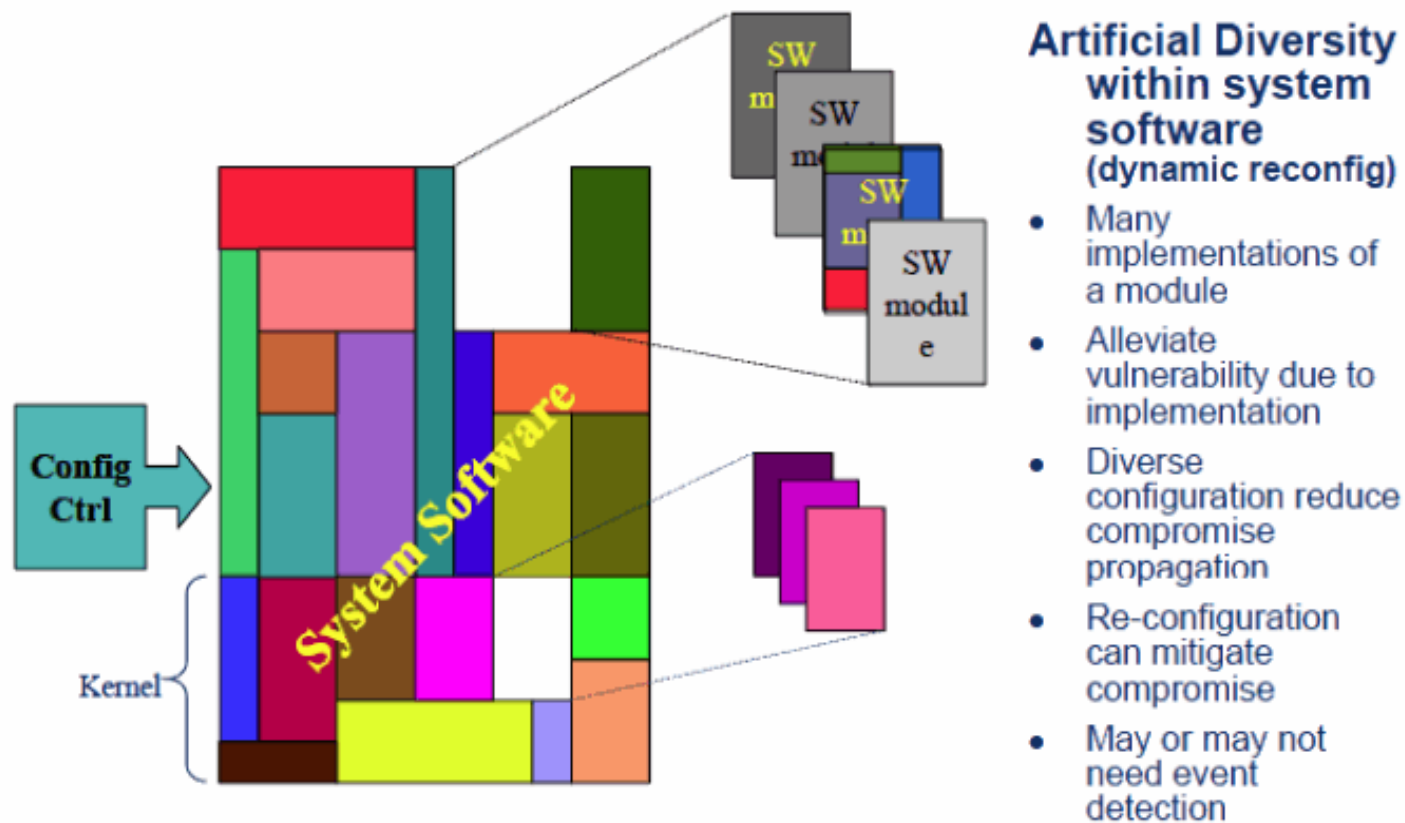
Conceptual architecture for Web-based command and control system (C2RPC)



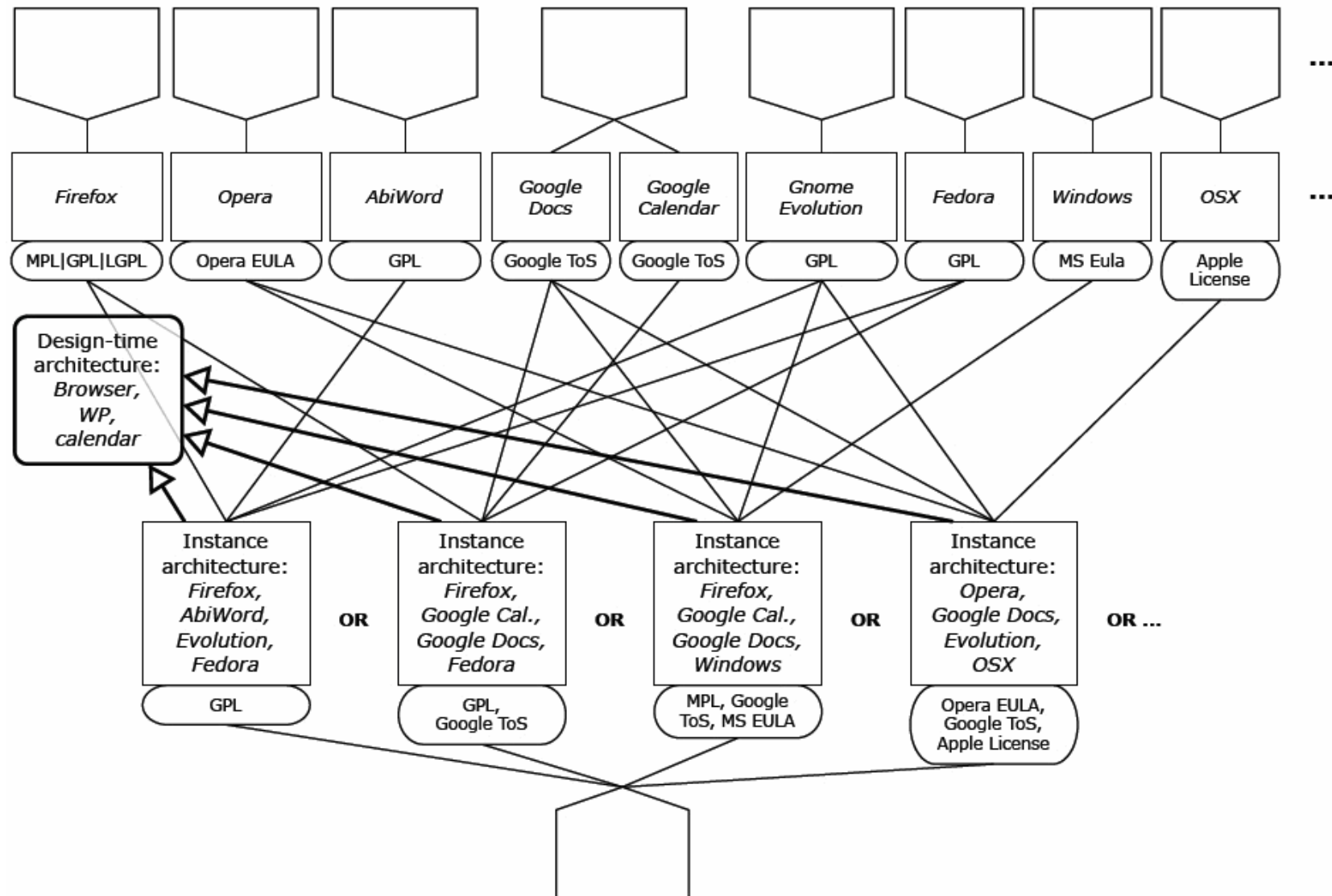
Design-time view of a C2RPC-like OA system



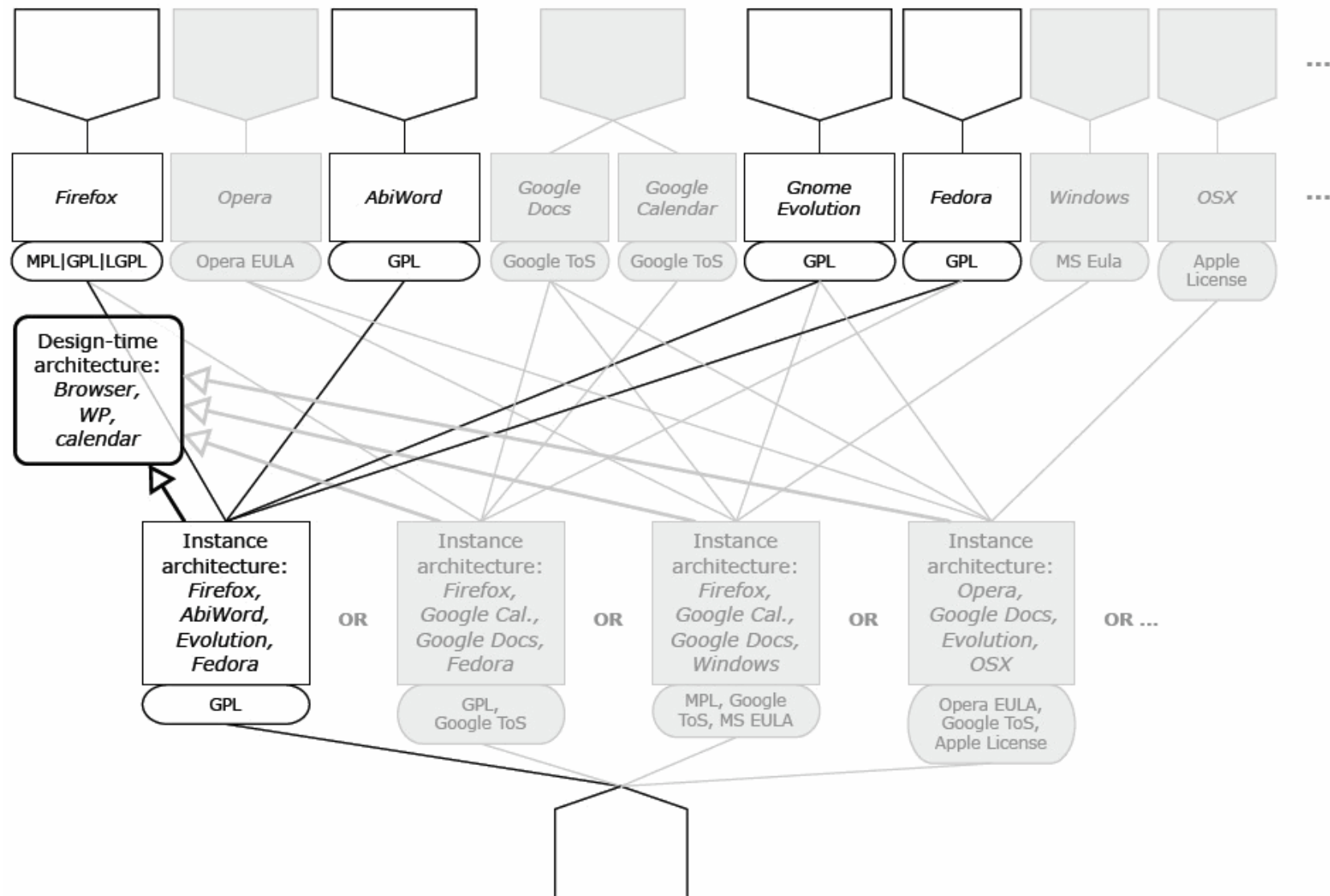
Software product line of *functionally equivalent* variants or *functionally similar* OA system versions enables support for artificial diversity and dynamic reconfiguration



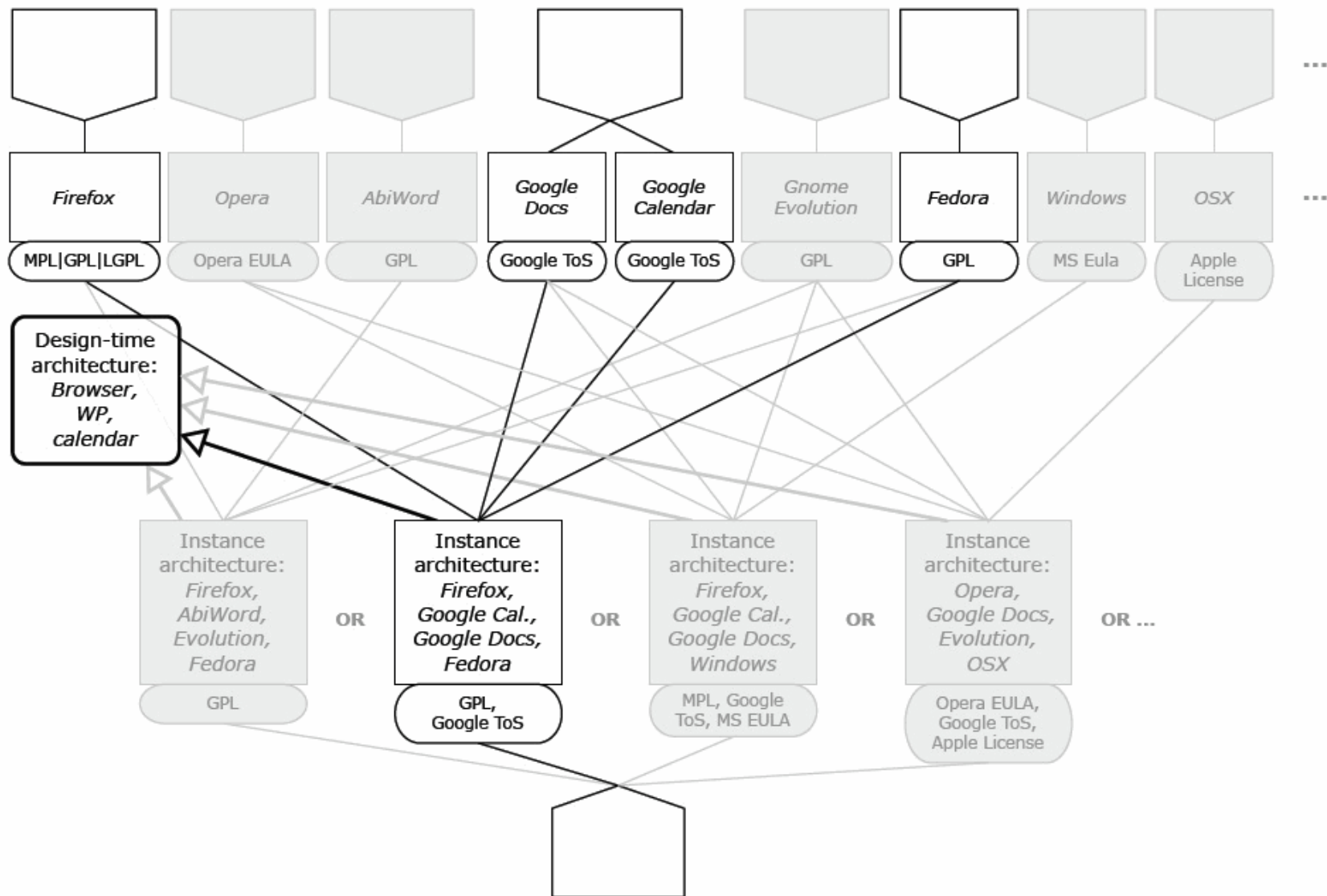
Software product line of *functionally similar* OA system alternatives



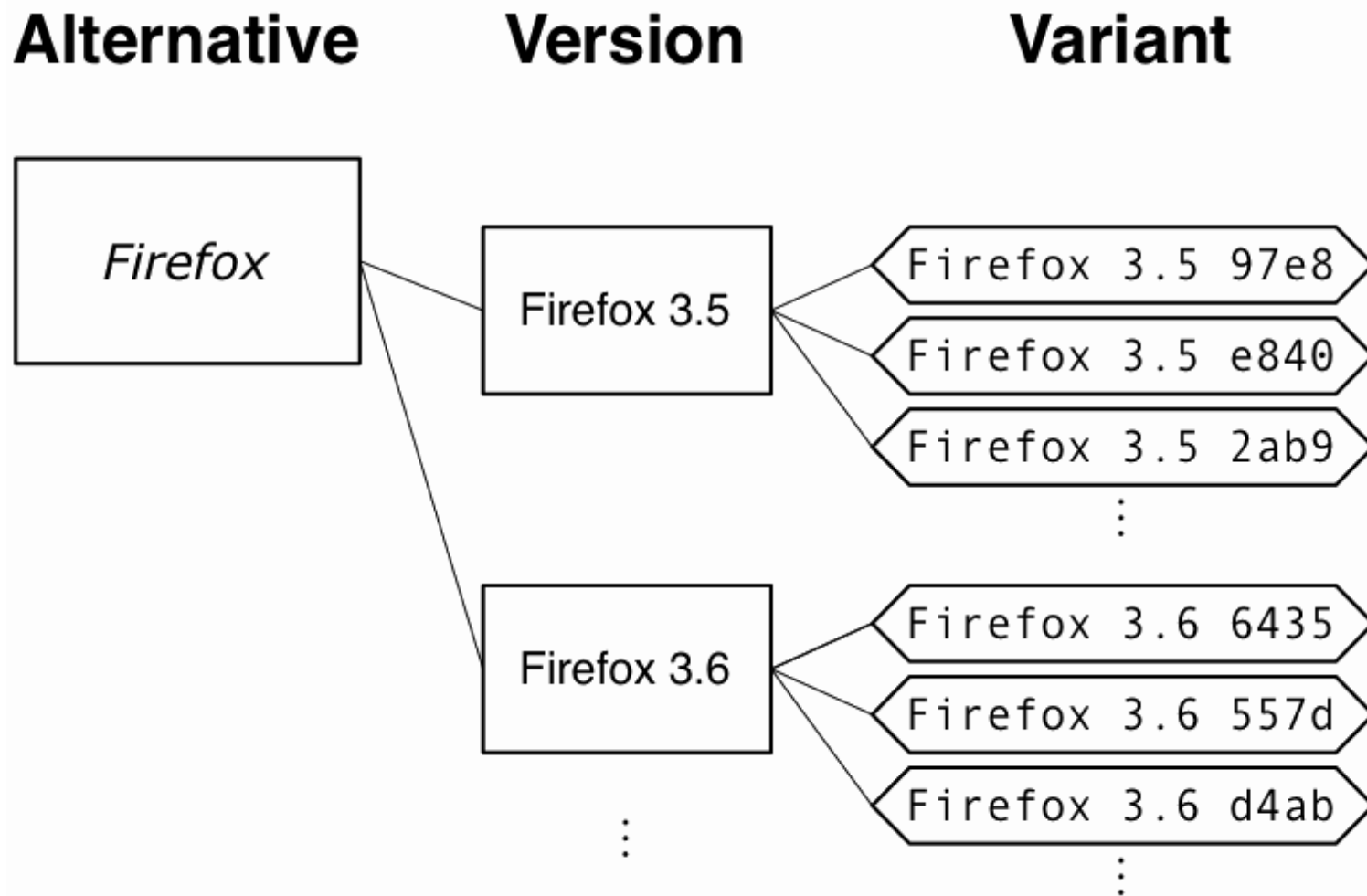
Product line selection of one alternative system configuration



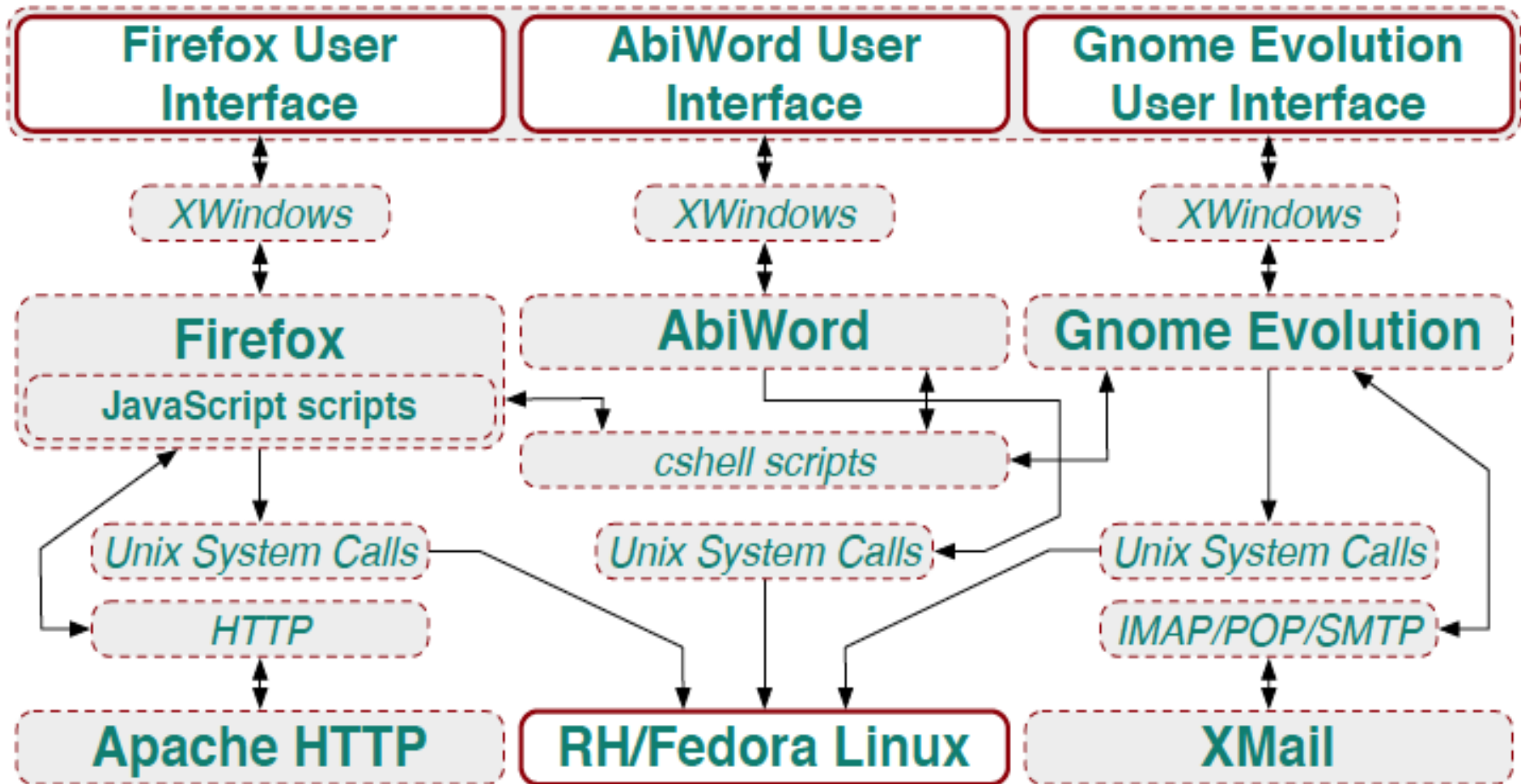
Product line selection of different functionally similar alternative



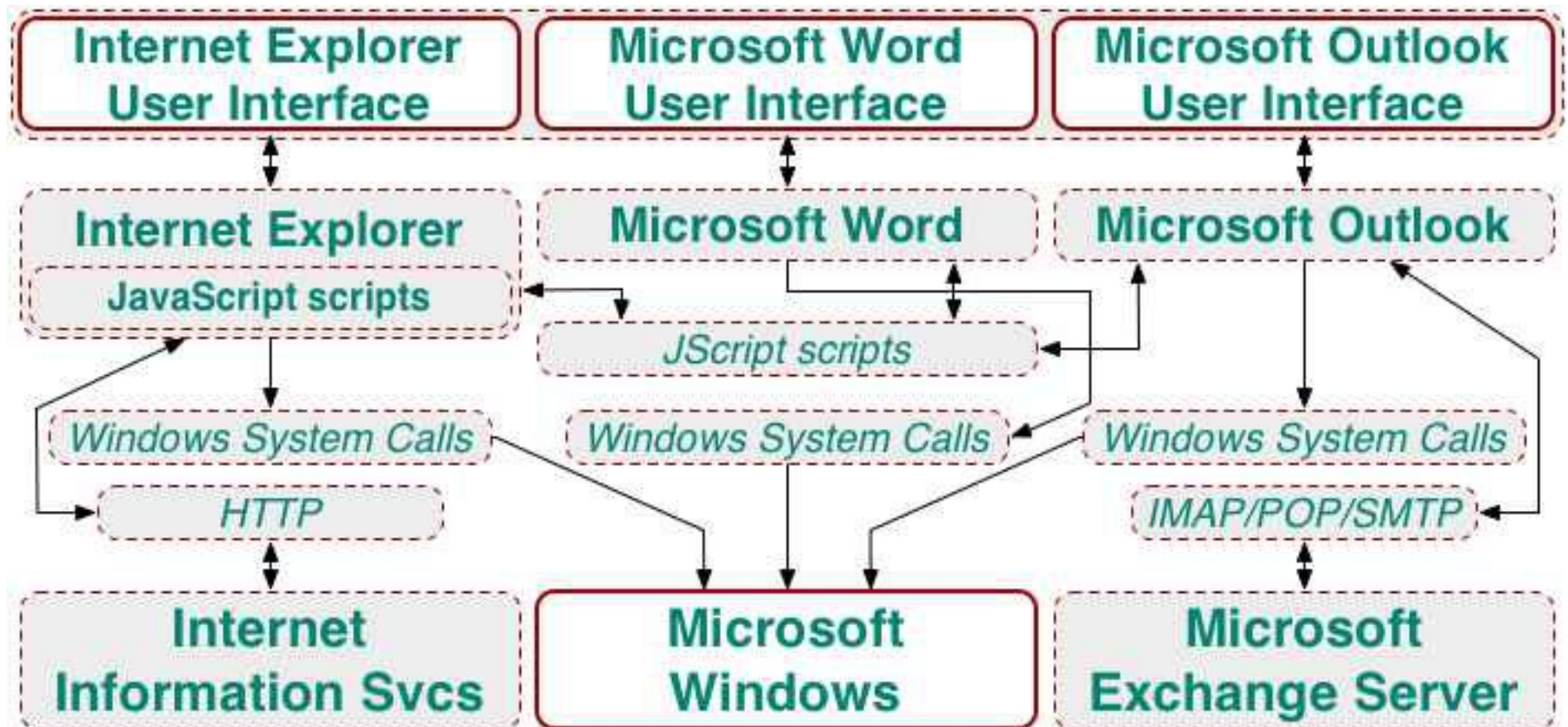
One Web browser component alternative, versions, and instance variants for inclusion via dynamic reconfiguration of OA system



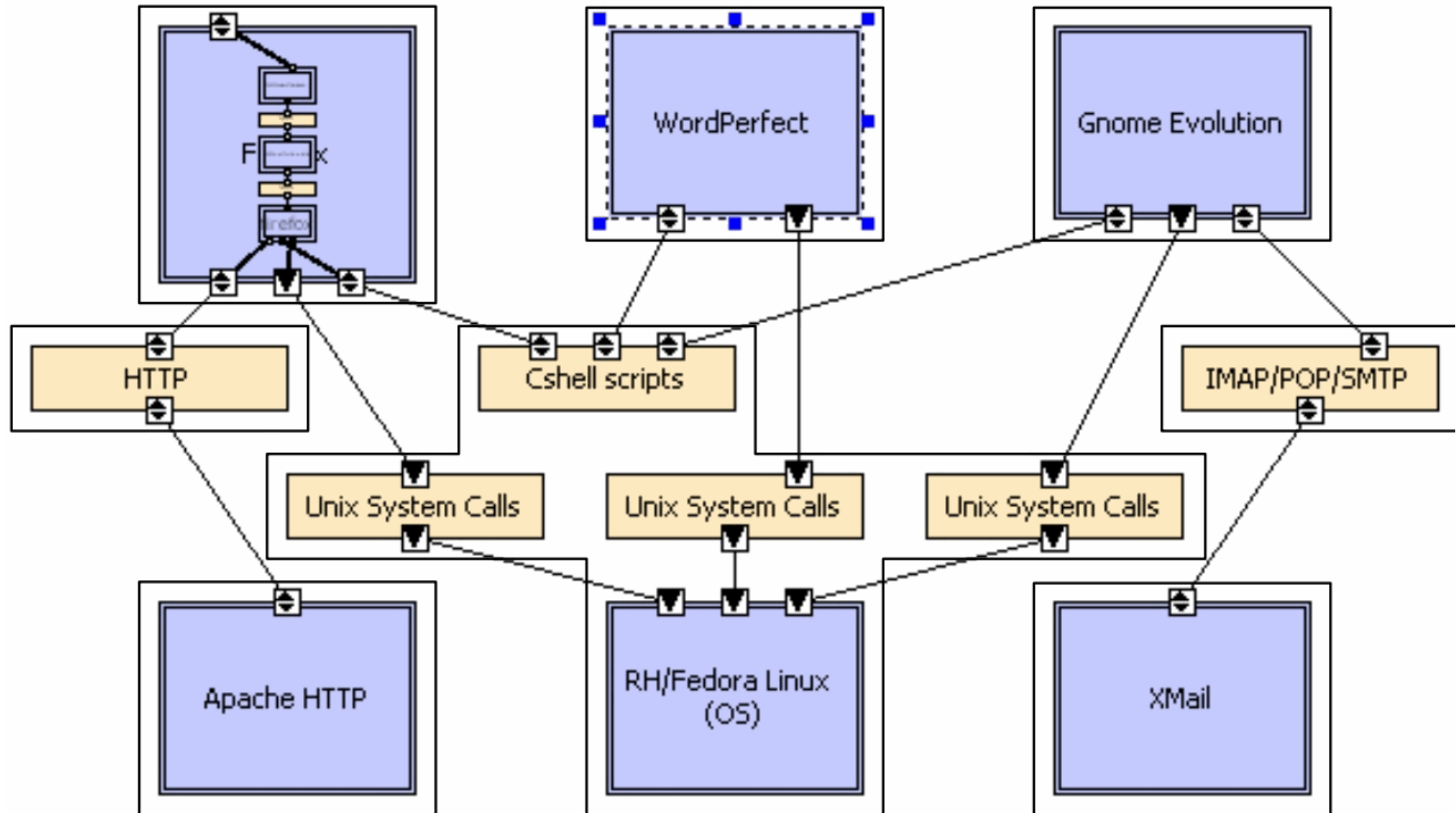
Build-time view of OA design selecting OSS product family alternatives



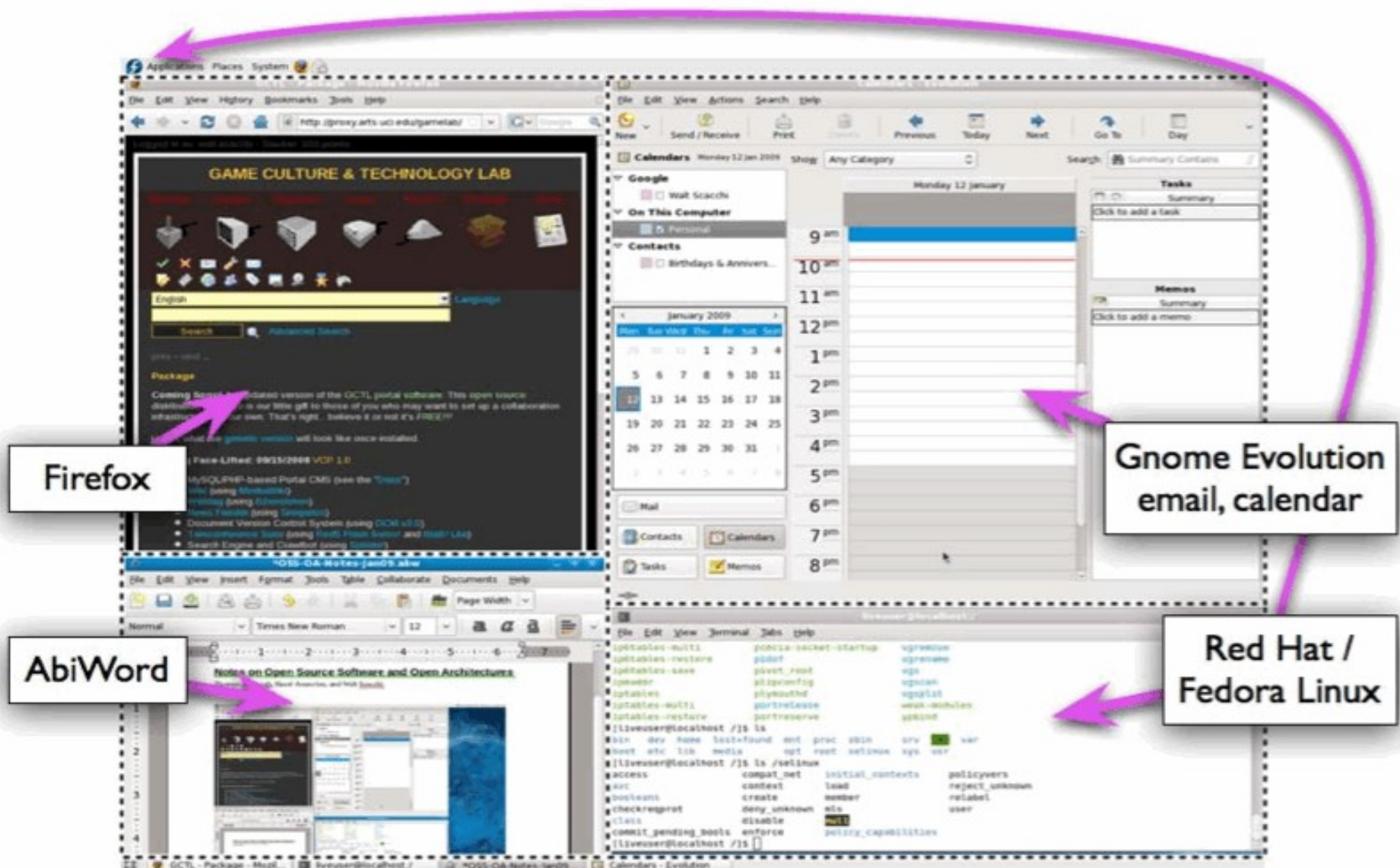
Build-time view of OA design selecting *proprietary* product family alternatives



Build-time view of an OA system security encapsulation scheme



Run-time deployment view of OA system family member configuration



The collage illustrates a workflow or process flow, indicated by a large pink arrow pointing from the top-left towards the bottom-right.

Top-Left Screenshot: A web browser window displaying the "GAME CULTURE & TECHNOLOGY LAB" website. The page features a dark background with various icons and text describing the lab's mission and approach. A pink arrow points to the "Search" button.

Top-Right Screenshot: A Google Docs document titled "A Composed Open Architecture Software System at Run-Time". The document is open in a web browser, showing the Google Docs interface and the document content. A pink arrow points to the "Share" button.

Bottom-Left Screenshot: A Google Calendar interface showing a weekly view for Monday, April 26, 2010. The calendar displays various events and a pink arrow points to the "Monday 4:00" slot.

Bottom-Right Screenshot: A terminal window showing system boot logs and user login information. The logs include messages from the "liveuser@localhost" system, such as "liveuser@localhost sbin\$ pwd", "liveuser@localhost sbin\$ cd ../selinux", and "liveuser@localhost selinux\$". A pink arrow points to the "liveuser@localhost selinux\$" prompt.

Firefox

Google Docs

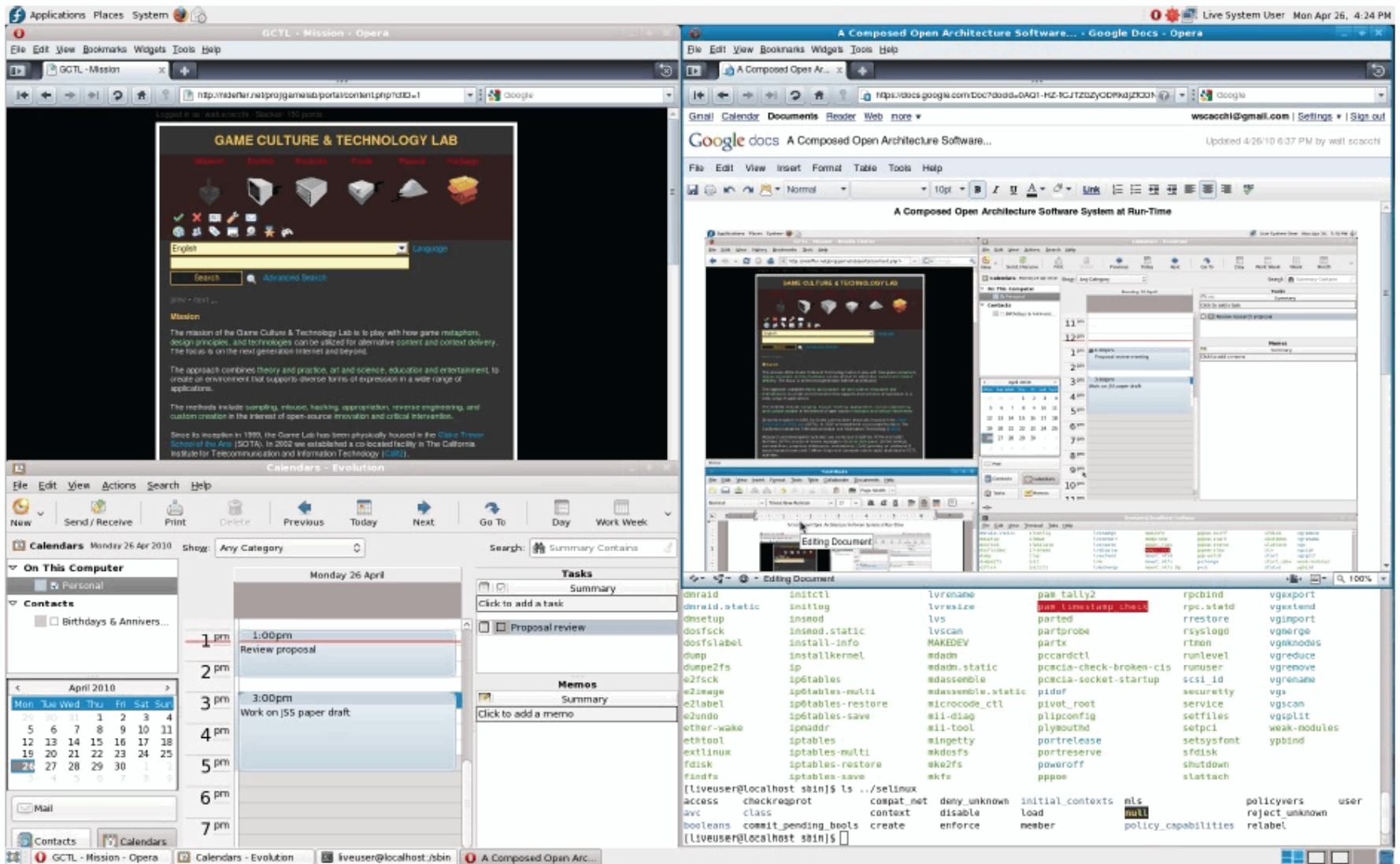
Red Hat /
Fedora Linux

Google
Calendar

Types of evolutionary changes in OA systems that also change system configurations

- Component (version) evolution
- Component replacement by similar alternative
- Architecture evolution
 - Including dynamic reconfiguration
- Component security policy license evolution
 - Licenses represent *annotated constraints* on OA components, connectors, and configurations
- Change in desired license rights or acceptable obligations within an OA system

Run-time deployment view with alternative OA configuration



Run-time deployment view with service-based OA configuration

The desktop environment displays the following windows:

- GCTL - Mission - Mozilla Firefox:** A web page for the Game Culture & Technology Lab. It features a navigation bar with links like 'Home', 'About', 'Services', 'Tools', 'Projects', and 'Contact'. The main content area includes a 'Mission' section and a 'Services' section.
- A Composed Open Architecture Software System at Run-Time - Google Docs - Mozilla Firefox:** A Google Docs document titled 'A Composed Open Architecture Software System at Run-Time'. The document content shows a screenshot of the GCTL website and a calendar view.
- Google Calendar - Mozilla Firefox:** A Google Calendar interface showing a calendar for April 2010. The calendar has a sidebar with 'My calendars' and 'Other calendars'.
- Terminal Window:** A terminal window titled 'liveuser@localhost:~\$' showing the output of the 'pwd' command, which is '/sbin'. The terminal also shows the output of the 'ls' command, listing various system files and directories.

Combinatorially large space of alternative OA system configurations

- Example: OA system with 6 component types
 - Each component type defines a family of functionally similar alternatives
 - Assume three overall system platform alternatives
 - Each component type with 5 alternative producers
 - Each alternative providing 5 available versions
 - Each version providing 5 functionally equivalent variants
- This allows for $6 \times 3 \times 5 \times 5 \times 5 = 2250$ similar but distinct OA system configurations available for at build-time.

Combinatorially large space of alternative OA system configurations

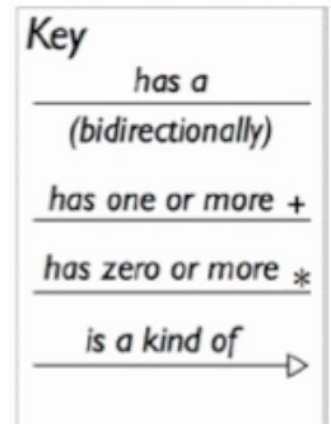
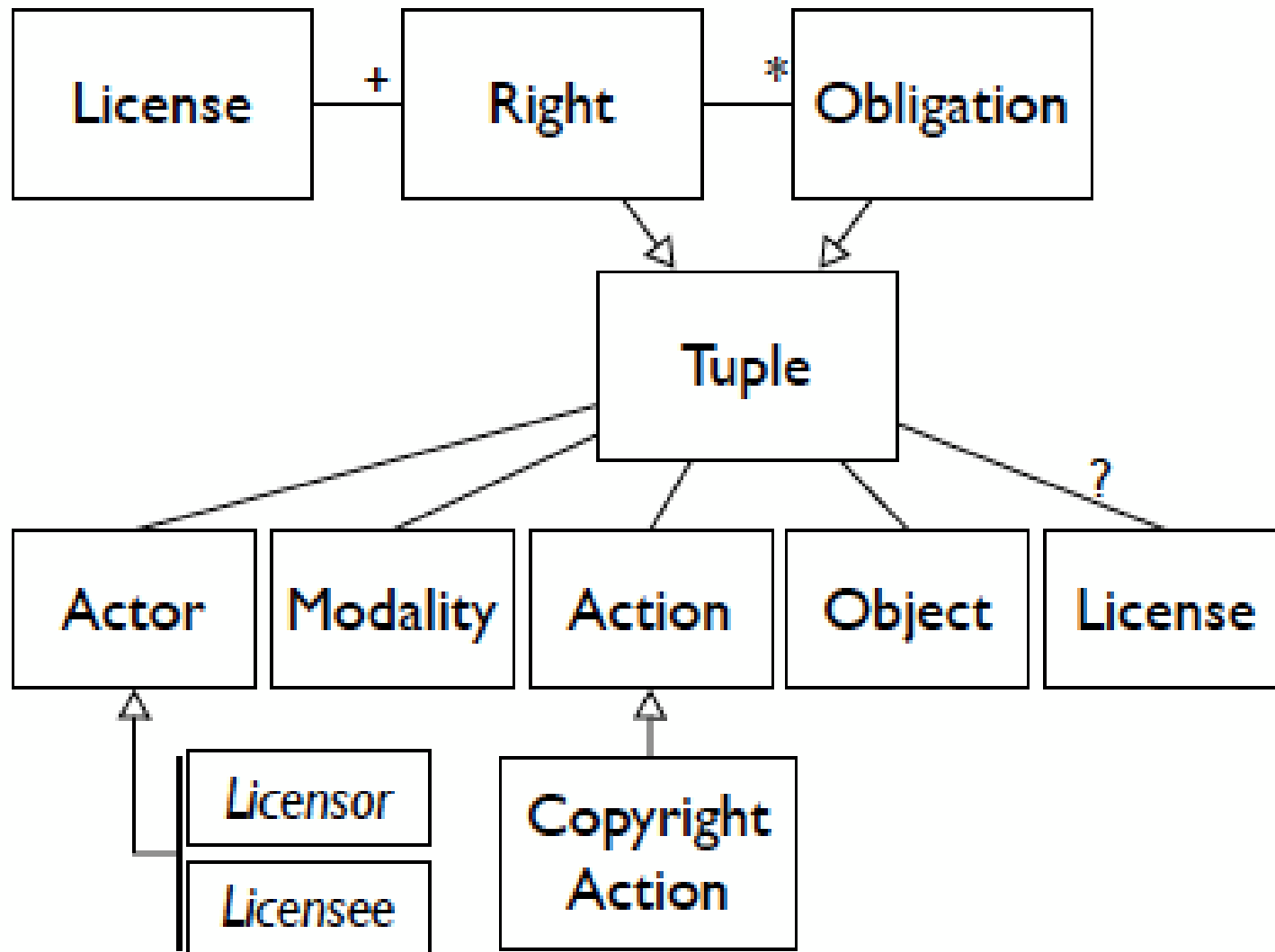
- Each configuration represents a specific attack surface, so *cost of attack grows combinatorially* with number of system alternatives to attack (i.e., which is the target now?)
- Cost of system defense via dynamic reconfiguration is low/constant
- Switching to alternative configurations can be handled via automated processes, driven by policy, e.g.:
 - switch run-time configuration every 30 minutes;
 - provide concurrent users with similar system configurations
 - monitor and continuously cross-check different user configurations for problem (attack or corrupted) operation
 - If configuration is problematic, then randomly switch to alternative similar configuration
 - If configuration is OK, then switch to equivalent alternative configuration at start of new usage sessions.

Software security licenses, architectures, and analysis

Specifying and analyzing system security requirements as “licenses”

- Security policies imply capabilities that correspond to “rights and obligations” in licenses
- Should be possible to specify and analyze *system security architecture* that conform to a *security meta-model*, much like we do for software licenses
- Should be possible to develop computational tools and development environments that can analyze security at design-time, build-time, and run-time, as well as when the system evolves

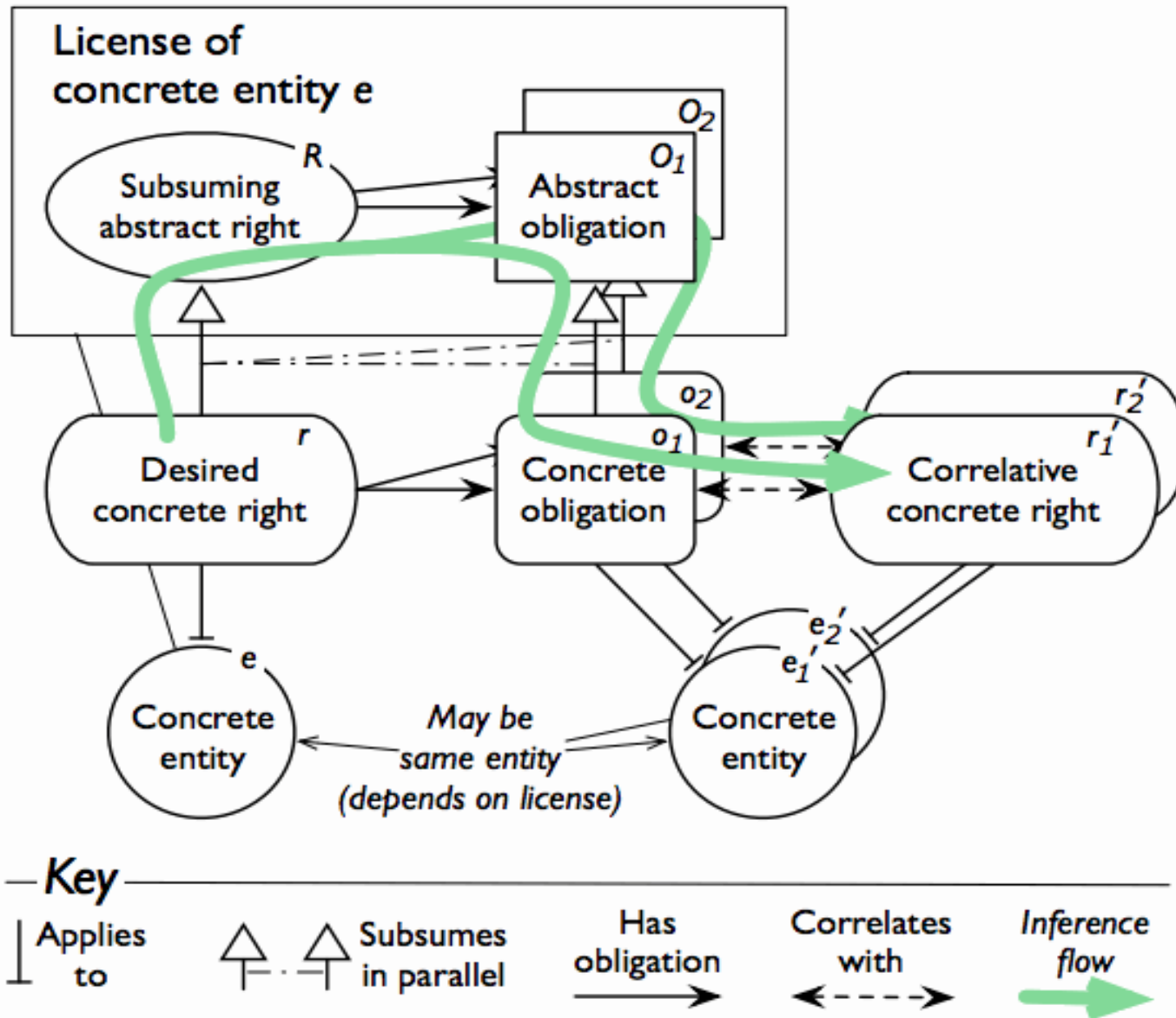
Software license meta-model for specifying constraint annotations



Logical modality and objects of software license rights and obligations constraints

	Actor	Modality	Action	Object	License (optional)	
Abstract Right	Licensee or Licensor	May or Need Not	The set of actions is large, comprising whatever actions the licenses in question utilize	Any Under This License	This License or Object's License	
				Any Source Under This License		
				Any Component Under This License		
Concrete Right				Concrete Object	Concrete License	
Concrete Obligation						
Abstract Obligation		Must or Must Not			Right's Object	Concrete License or Right's License
					All Sources Of Right's Object	
	X Scope Sources					
	X Scope Components					

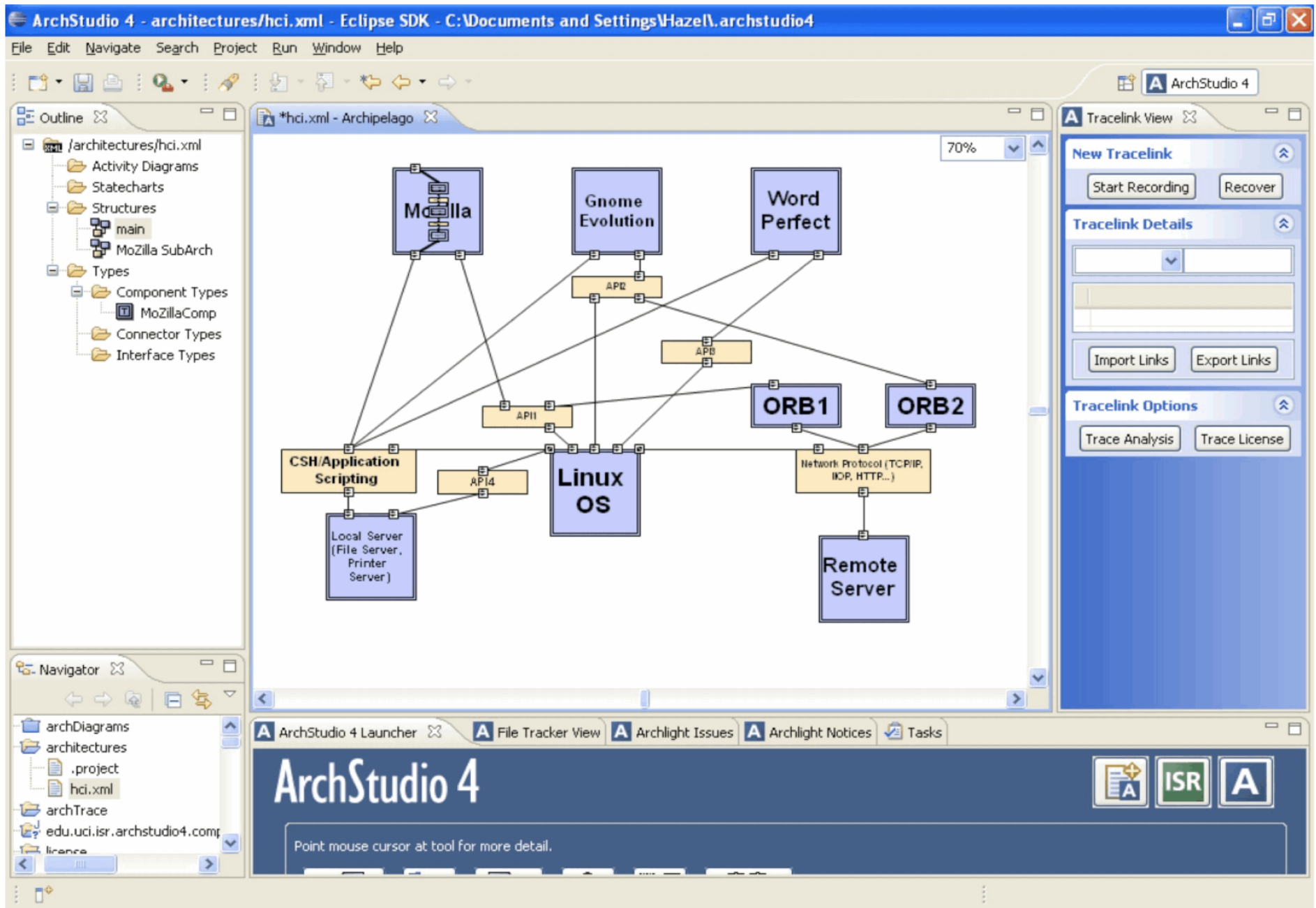
License inference scheme



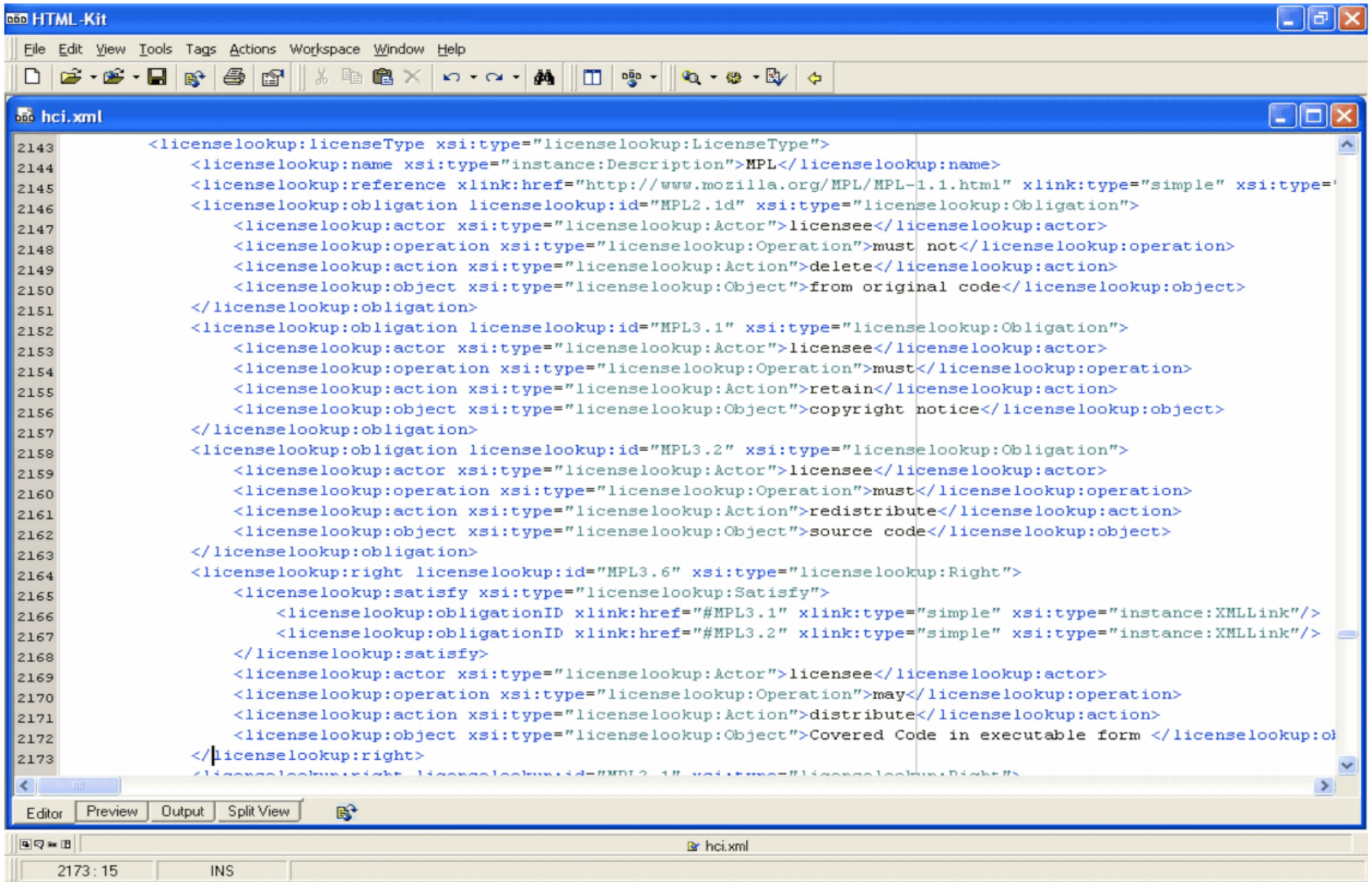
Security license analysis

- License types:
 - Strongly reciprocal, weakly reciprocal, academic, Terms of Service, Proprietary
- Propagation of reciprocal obligations
- Conflicting obligations
- Calculating obligations and rights

Prototype view of OA system development environment with license analysis plug-in



Internal form of component license annotation of OA prior to analysis



The screenshot shows the HTML-Kit editor with the file hci.xml open. The editor displays XML code for license annotations. The code is structured as follows:

```
<licenselookup:licenseType xsi:type="licenselookup:LicenseType">
  <licenselookup:name xsi:type="instance:Description">MPL</licenselookup:name>
  <licenselookup:reference xlink:href="http://www.mozilla.org/MPL/MPL-1.1.html" xlink:type="simple" xsi:type="instance:XMLLink"/>
  <licenselookup:obligation licenselookup:id="MPL2.1d" xsi:type="licenselookup:Obligation">
    <licenselookup:actor xsi:type="licenselookup:Actor">licensee</licenselookup:actor>
    <licenselookup:operation xsi:type="licenselookup:Operation">must not</licenselookup:operation>
    <licenselookup:action xsi:type="licenselookup:Action">delete</licenselookup:action>
    <licenselookup:object xsi:type="licenselookup:Object">from original code</licenselookup:object>
  </licenselookup:obligation>
  <licenselookup:obligation licenselookup:id="MPL3.1" xsi:type="licenselookup:Obligation">
    <licenselookup:actor xsi:type="licenselookup:Actor">licensee</licenselookup:actor>
    <licenselookup:operation xsi:type="licenselookup:Operation">must</licenselookup:operation>
    <licenselookup:action xsi:type="licenselookup:Action">retain</licenselookup:action>
    <licenselookup:object xsi:type="licenselookup:Object">copyright notice</licenselookup:object>
  </licenselookup:obligation>
  <licenselookup:obligation licenselookup:id="MPL3.2" xsi:type="licenselookup:Obligation">
    <licenselookup:actor xsi:type="licenselookup:Actor">licensee</licenselookup:actor>
    <licenselookup:operation xsi:type="licenselookup:Operation">must</licenselookup:operation>
    <licenselookup:action xsi:type="licenselookup:Action">redistribute</licenselookup:action>
    <licenselookup:object xsi:type="licenselookup:Object">source code</licenselookup:object>
  </licenselookup:obligation>
  <licenselookup:right licenselookup:id="MPL3.6" xsi:type="licenselookup:Right">
    <licenselookup:satisfy xsi:type="licenselookup:Satisfy">
      <licenselookup:obligationID xlink:href="#MPL3.1" xlink:type="simple" xsi:type="instance:XMLLink"/>
      <licenselookup:obligationID xlink:href="#MPL3.2" xlink:type="simple" xsi:type="instance:XMLLink"/>
    </licenselookup:satisfy>
    <licenselookup:actor xsi:type="licenselookup:Actor">licensee</licenselookup:actor>
    <licenselookup:operation xsi:type="licenselookup:Operation">may</licenselookup:operation>
    <licenselookup:action xsi:type="licenselookup:Action">distribute</licenselookup:action>
    <licenselookup:object xsi:type="licenselookup:Object">Covered Code in executable form</licenselookup:object>
  </licenselookup:right>
  <licenselookup:right licenselookup:id="MPL3.1" xsi:type="licenselookup:Right">
```

The status bar at the bottom shows the cursor position at line 2173, column 15, and the current mode is INS (Insert).

Directory of computational methods for analyzing “rights”

RightConcrete (Thomas' Packages)

file:///Users/thomas/java/javadoc/index.html

Reader

All Classes

Packages

- [automaton](#)
- [bibtex](#)
- [commands](#)
- [eventmatch](#)
- [fologic](#)
- [lambda](#)
- [license0](#)
- [license3](#)
- [nesting](#)
- [org.thomasalebaugh.command](#)

Classes

- [Action](#)
- [Actor](#)
- [Calculation](#)
- [Command](#)
- [Domain](#)
- [DomainScope](#)
- [DomainSet](#)
- [FormulaConjunction](#)
- [FormulaDisjunction](#)
- [LicenseAbstract](#)
- [LicenseConcrete](#)
- [LicenseConcretePrecursor](#)
- [LicenseQuantifier](#)
- [LicenseReference](#)
- [LicenseSet](#)
- [LicenseVirtual](#)
- [Modality](#)
- [ModalityObligation](#)
- [ModalityRight](#)
- [ObligationAbstract](#)
- [ObligationConcrete](#)

Method Summary

Action	action() The right's action.
Actor	actor() The right's actor.
int	compareTo(Named _o) Compares this object with the specified object in a total ordering by <code>name()</code> .
boolean	conflictsWith(ObligationAbstract _oa) True if this right conflicts with <code>_obligation</code> .
ObligationConcrete	correlative() The correlative obligation.
static RightConcrete	factory(Actor _actor, ModalityRight _modality, Action _action, PatientConcrete _patient, LicenseConcrete _license) Returns the concrete right that has the given arguments, constructing it if necessary.
static RightConcrete	factory(RightAbstract _abstractRight, PatientConcrete _context) Returns a concrete right subsuming an abstract right.
LicenseConcrete	license() The right's license argument.
Modality	modality() The right's modality (a ModalityRight).
String	name() The name (same as <code>toString()</code>) used in comparisons.
protected boolean	overlapsTuple(License3.Tuple _b) True iff subsumes(<code>_b</code>) OR <code>_b.subsumes(this)</code> .
PatientConcrete	patient() The right's patient argument.
boolean	reserved() True if this right overlaps with a reserved right.
RightAbstract	reservedRight()

License review during license analysis

The screenshot displays the ArchStudio 4 IDE interface. The main workspace shows a project diagram with components like Firefox, WordProcessor, and GnomeEvolution, connected to various protocols and systems. The right sidebar contains a 'View License Information' panel for the 'WordProcessor' component, showing details like 'License: GPL 2', 'Code: WordProcessor', and '1st Rev #: 9500'. Below this, there's an 'Inference Tree' and 'Options' section. The bottom of the screen shows a web browser window displaying the 'GNU General Public License Version 2, June 1991'. The browser window includes a sidebar with a list of licenses (AFL 3.0, AGPL v.3, Apache 2.0, etc.) and a main content area with the license text, including sections like 'Preamble' and 'COPYING, DISTRIBUTION AND MODIFICATION'.

ArchStudio 4 - HLS_Project/src/arch/hci.xml - Eclipse SDK - C:\Users\Hazel\AS_testWorkspace

File Edit Navigate Search Project Run Window Help

Outline

- file:/C:/Users/Hazel/AS_testWorkspace/H
- Activity Diagrams
- Statecharts
- Structures
- main
- Types
- Component Types
- WordProcessor_v1
- Connector Types
- Interface Types

hci.xml - Archipelago

100%

Firefox

WordProcessor

GnomeEvolution

HTTP

Cshell scripts

IMAP/POP/SMTP

Unix System

Unix System

Unix System

View License Information

License: GPL 2 View

Code: WordProcessor View

1st Rev #: 9500

1st License: CTL

Latest Rev #: 9504

Inference Tree

Options

License Analysis License Preferences

GPLv2 numbered - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://www.cs.georgetown.edu/~alspaugh/pub/osl-sps/gpl2.0.html

Yahoo! Search

GNU General Public License Version 2, June 1991

HOME

FOSSIL §1 home

FOSSIL stats

1 AFL 3.0

2 AGPL v.3

3 Apache 2.0

4 Artistic 2.0

5 BSD Template

6 (BSD 4-clause)

7 CDDL 1.0

8 CPL 1.0

9 CPOL 1.02

10 EPL 1.0

11 GPL v2.0

12 GPL v3.0

13 LGPL v2.1

14 LGPL v3.0

15 MIT

16 MPL 1.0

17 MPL 1.1

18 Ms-PL

19 OS 3.0

20 PHP 3.0

21 Ruby

22 zlib/libpng

23 CTL

Under construction:

24 CC-BY-SA3.0

Source: OSI text (2009Jul07)

Hide/show §1s Hide/show notes

Program work based on the Program modification you

Preamble

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. NO WARRANTY 11. 12. How to Apply These Terms to Your New Programs

§1s1 Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

§2s1 Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

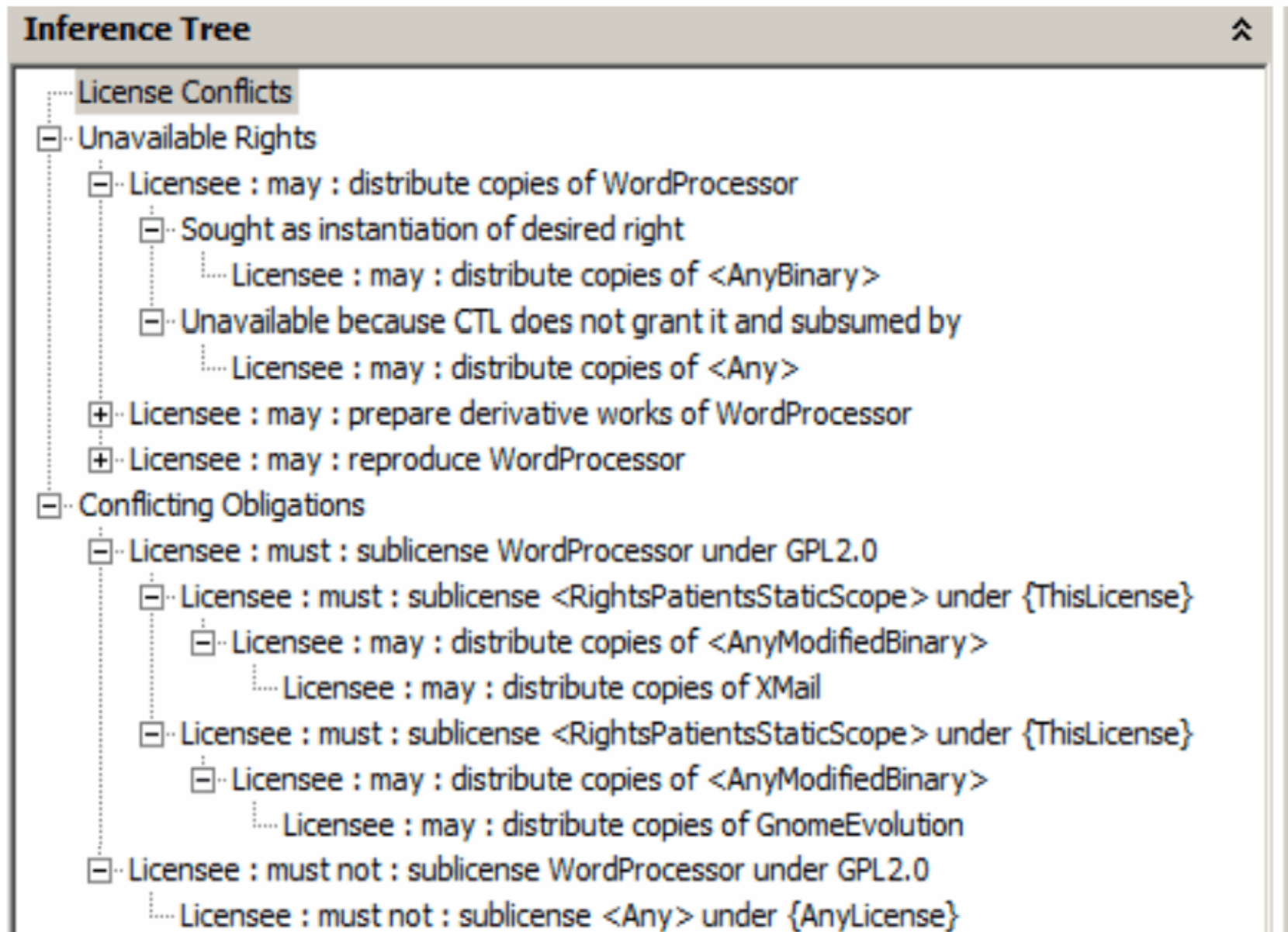
§1s1 Preamble

§1s1s1 The licenses for most software are designed to take away your freedom to share and change it. §1s1s2 By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. §1s1s3 This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. §1s1s4 (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) §1s1s5 You can apply it to your programs, too.

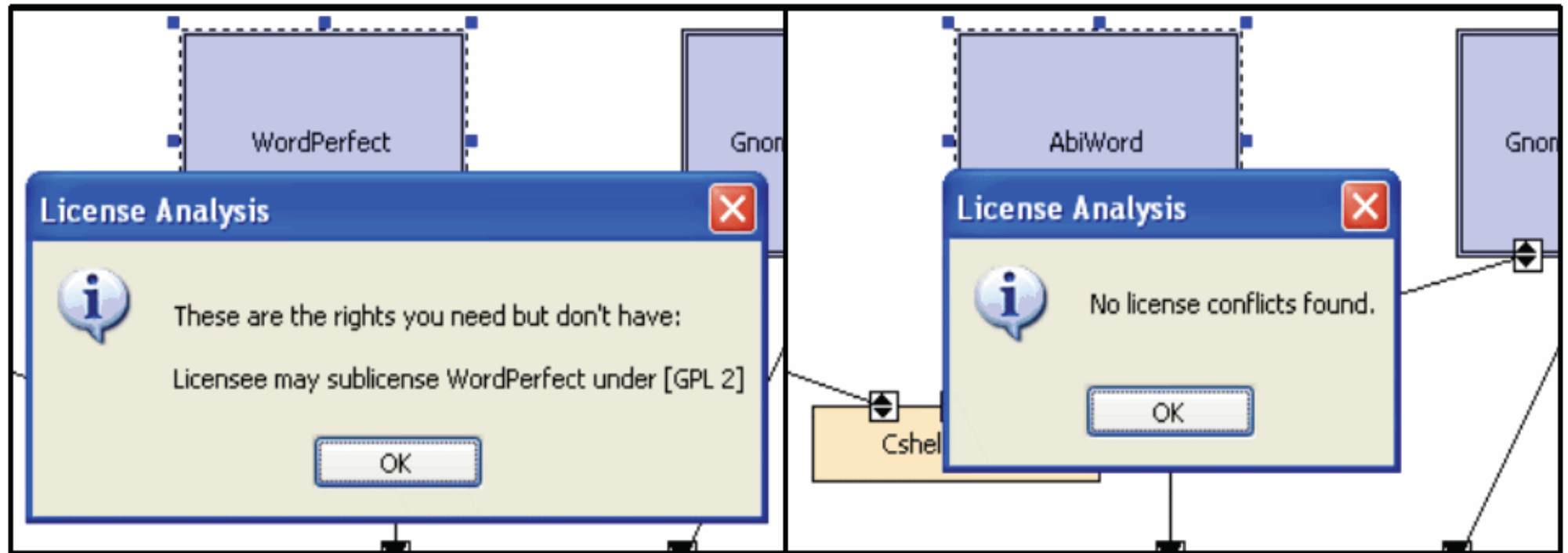
§1s2s1 When we speak of free software, we are referring to freedom, not price. §1s2s2 Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; §2.1 and that you know you can do these things.

§1s3s1 To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. §1s3s2 These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it

Reasoning structure during analysis



Results from license analyses with system component replacement



Discussion

- Cyber security challenges of interest
 - Automated reasoning about OA system security via robust and resilient autonomic host services
 - System self-awareness
 - Self-diagnosis (static/dynamic monitoring)
 - Self-healing (dynamic OA reconfiguration)
 - Supporting graceful degradation and artificial diversity
- Reasoning about secure Open Architecture system obligations and rights
- Examples of recent or work-in-progress results

Acknowledgements

This presentation was developed under work supported by the Naval Postgraduate School Acquisition Research Program Assistance Agreement No. #N00244-10-1-0038, #N00244-12-1-0004 and #N00244-12-1-0067 awarded by the U.S. Fleet Industrial Supply Center, San Diego (FISC-SD). The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. FISC-SD and NPS. The FISC-SD and NPS do not endorse any products or commercial services mentioned in this publication.

Funding also from grants #0808783 and #1256593 from the National Science Foundation. No review, approval, or endorsement implied.