

Supporting Reconfigurable Object Distribution for Customized Web Applications

Po-Hao Chang

RiverGlass, Inc.

Gul Agha

University of Illinois at Urbana-Champaign

Outline

- Introduction
- Virtual Framework
- Specification Rules
- Reconfiguring Distribution
- Related Work
- Discussion and Conclusion

Outline

- **Introduction**
- Virtual Framework
- Specification Rules
- Reconfiguring Distribution
- Related Work
- Discussion and Conclusion

Motivation

- Web applications are gaining popularity
 - *Universal Availability* (HTML + HTTP)
- Universal good applications?
 - Context assumptions required
 - ▶ Thin clients? Fat clients?
 - ▶ Broadband? Dial-up? 3G?
- Consider a Web application is a composition of objects
 - *Location*: Where to run? Where to create?
 - *Timing*: When to load? How much to load?

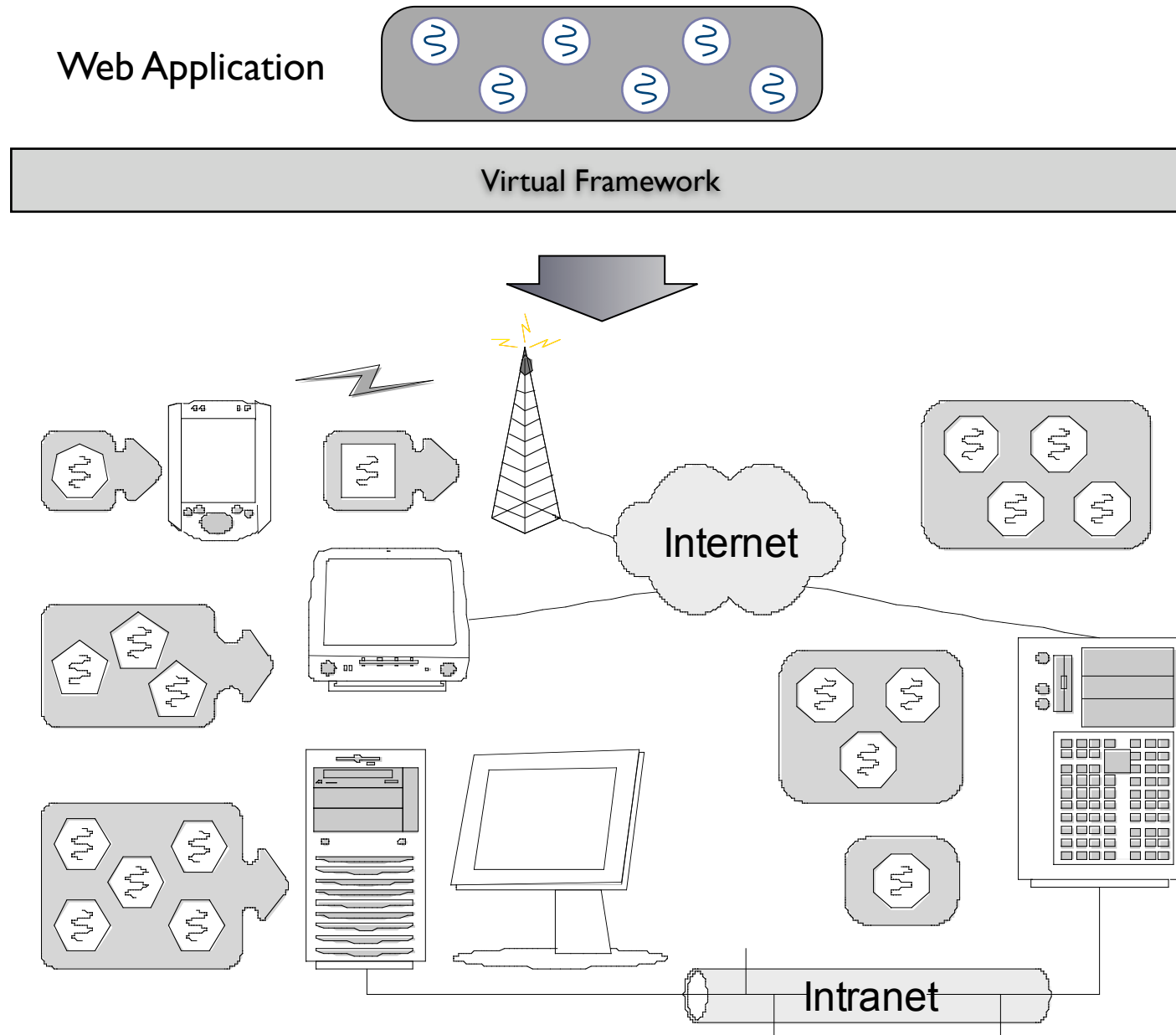
Approach

- Enable customizable Web applications
 - Web application = product line = composition of objects
 - Adaptable objects
 - Reconfigurable object distribution
- Strategies
 - Virtualization
 - Separation of concerns
 - Generative components

Outline

- Introduction
- **Virtual Framework**
- Specification Rules
- Reconfiguring Distribution
- Related Work
- Discussion and Conclusion

Virtualization



Actor Model

- Requirements:
 - Component-based model
 - No shared information
 - Communication only through message exchange
 - Can express asynchronous interaction
- ActorScript = JavaScript - Object + Actor
 - Self-contained
 - Asynchronous method invocation
 - Created by initial behavior (prototype)
- Application = a set of ActorScript prototypes

Outline

- Introduction
- Virtual Framework
- **Specification Rules**
- Generative Application Framework
- Related Work
- Discussion and Conclusion

Actor Annotation

- Specification = actor annotation
 - Intuitive concerns: location, language ...
 - Sophisticated concerns: moving policy, redundancy
- Problem: how to select an actor?
 - No runtime info: pointer value? reference? unique id?
 - [actor feature] : attribute = value
 - Annotate on a set of actors

Selection by Prototype

- Actor feature: Actor's behavior = prototype name
- Prototype specification
 - [prototype name] : attribute = value
 - DateValidator : Location = Client
- Same as class-level annotation
 - Easy to implement
 - Not detailed enough

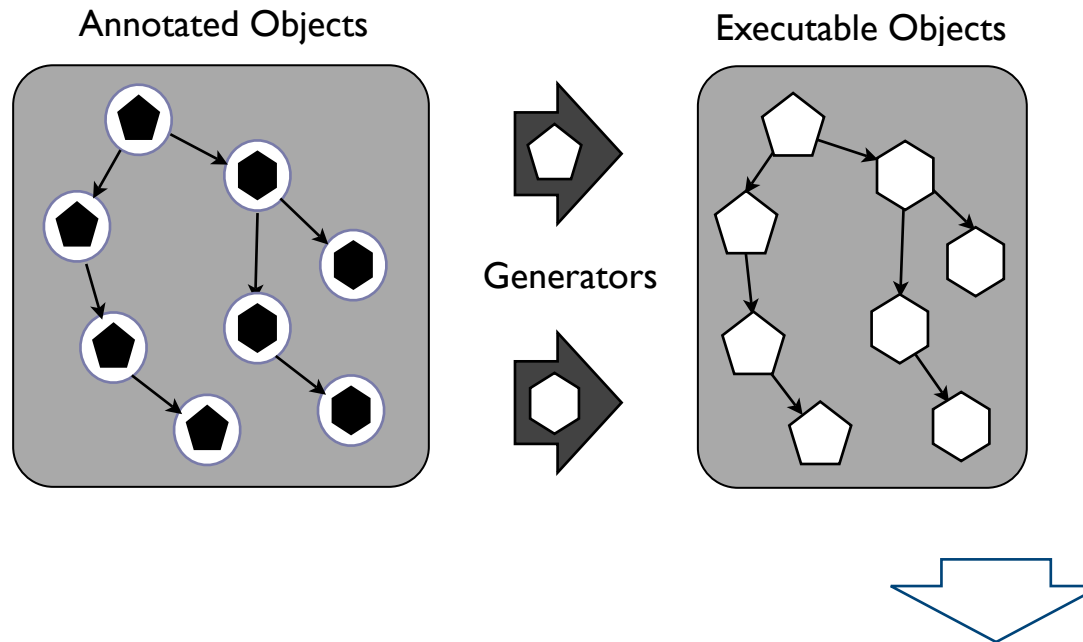
Selection by Genealogy

- If a prototype is used by more than one prototypes ...
 - The actors of the prototype can be classified further
- Genealogy specification
 - [prototype X] > [prototype Y] : attribute = value
 - OrderForm > Button : Location = Server
 - InventoryForm > Button : Location = Client
 - [prototype X] > [prototype Y] > [prototype Z] ...
- More specific rule overrides
 - No conflict: genealogy ordering is linear

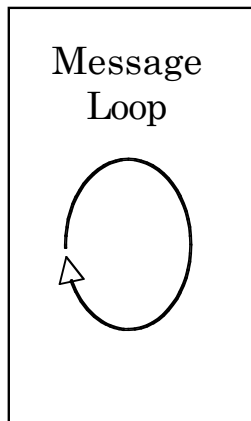
Outline

- Introduction
- Virtual Framework
- Specification Rules
- **Reconfiguring Distribution**
- Related Work
- Discussion and Conclusion

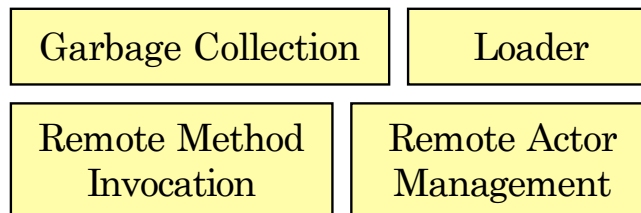
Generative Components



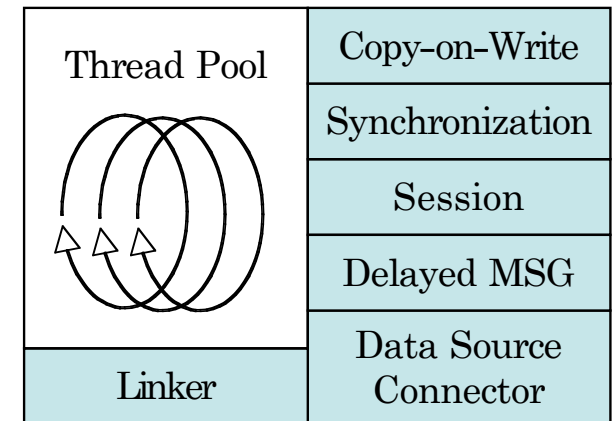
Client Podium



Execution Framework



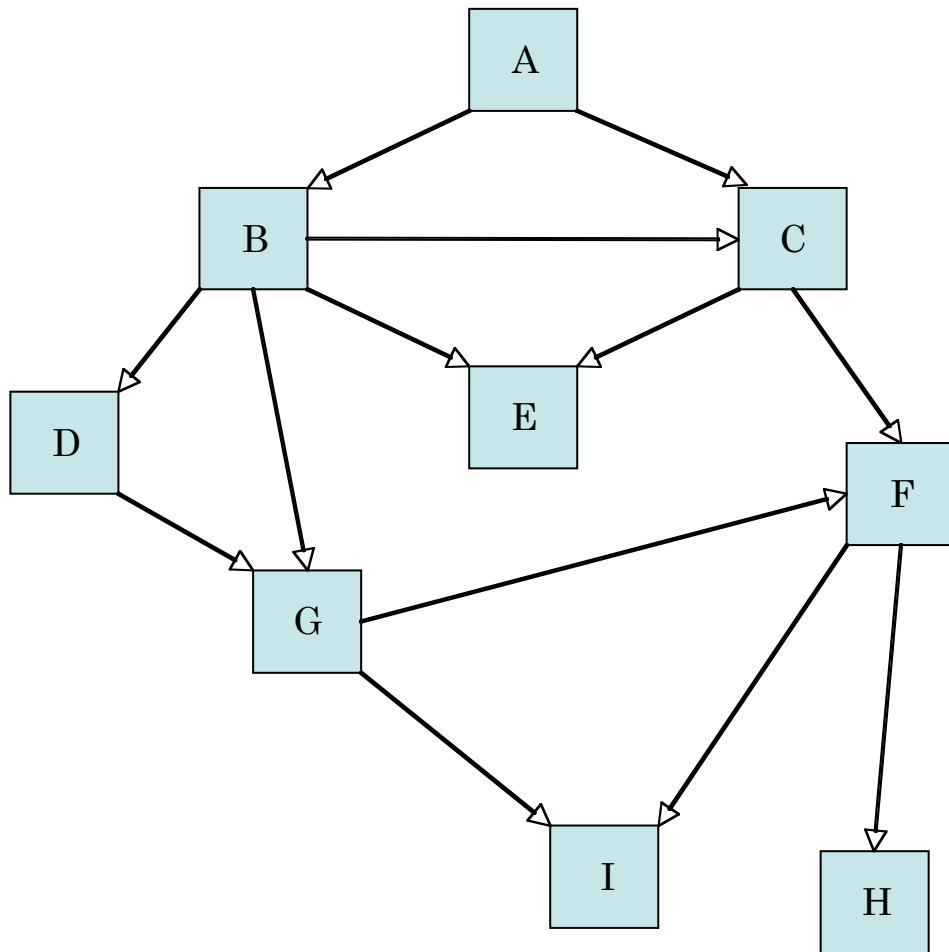
Server Podium



Algorithm

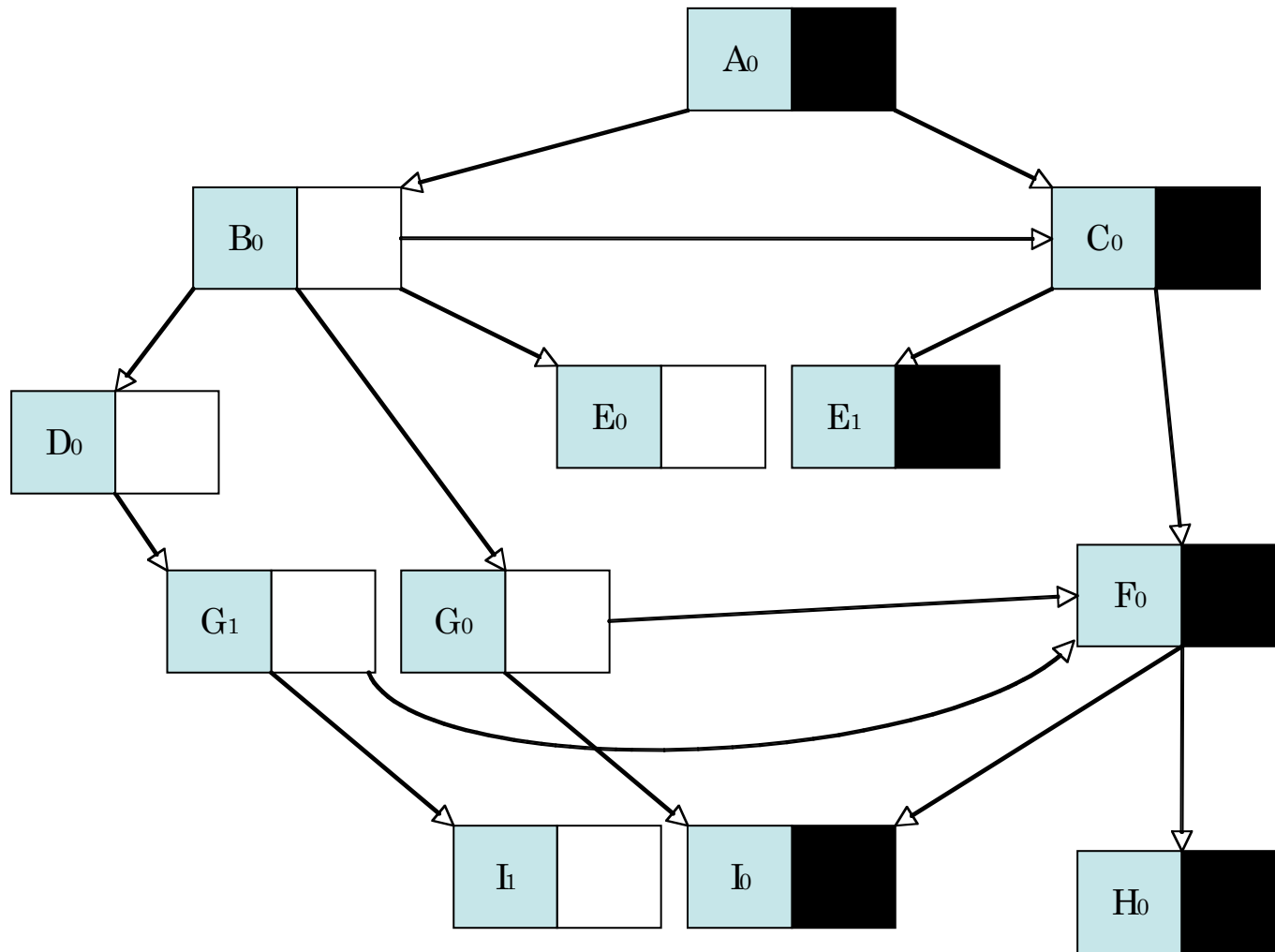
- Input: Application + Specification
 - Prototype Dependency Graph (PDG) + Rules
- Output: Customized Application
 - Implementation Linkage Graph (ILG)
- Assumptions:
 - One attribute
 - Default implementation

PDG + Rules

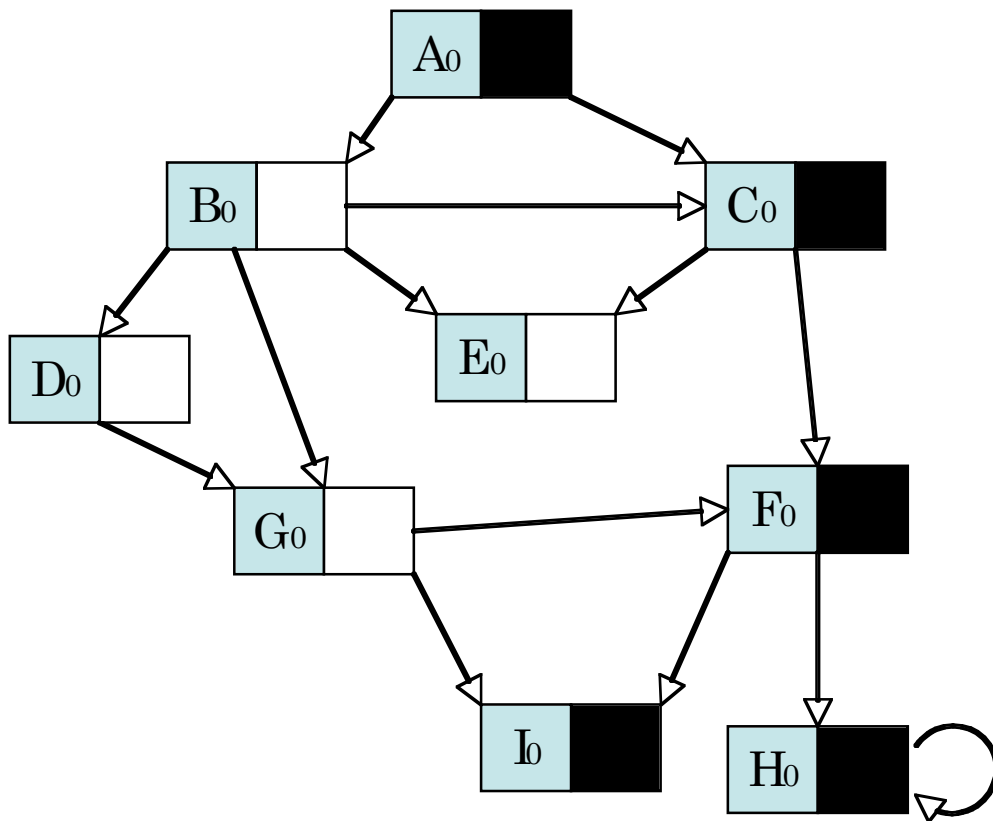


1. A : Attr = Blk;
2. B : Attr = Wht;
3. C : Attr = Blk;
4. D : Attr = Wht;
5. E : Attr = Wht;
6. F : Attr = Blk;
7. G : Attr = Wht;
8. H : Attr = Blk;
9. I : Attr = Blk;
10. C > E : Attr = Blk;
11. D > G > I : Attr = Wht;

Implementation Linkage Graph



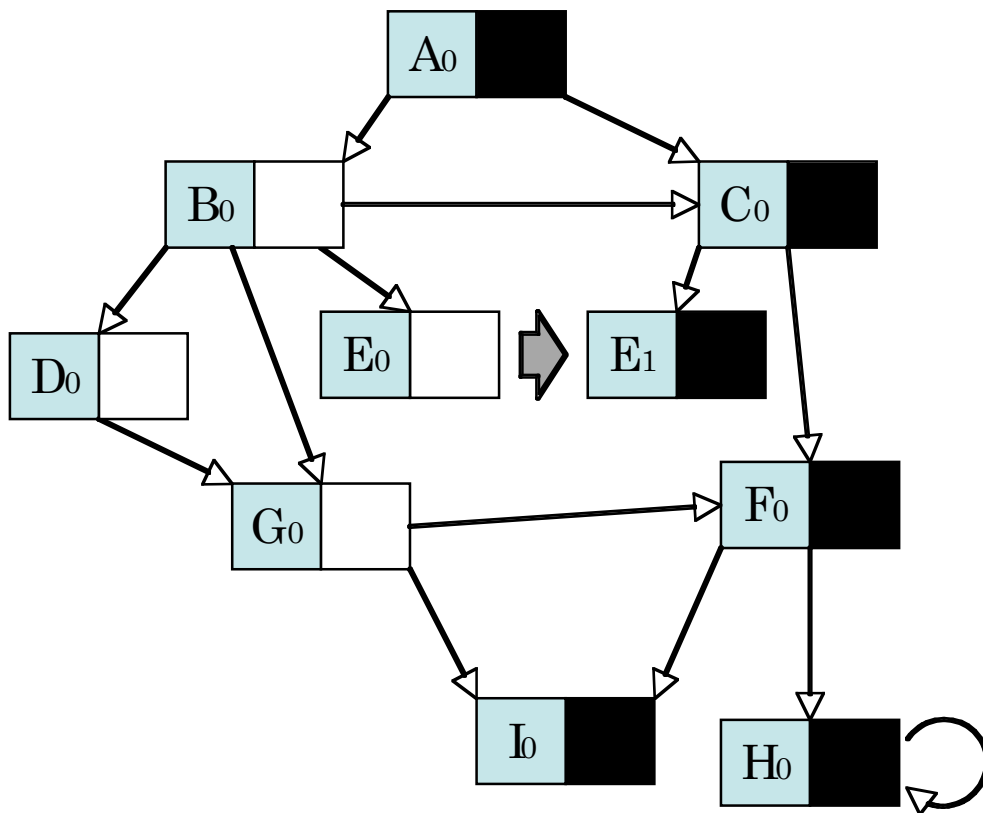
Default Implementations



1. A : Attr = Blk;
2. B : Attr = Wht;
3. C : Attr = Blk;
4. D : Attr = Wht;
5. E : Attr = Wht;
6. F : Attr = Blk;
7. G : Attr = Wht;
8. H : Attr = Blk;
9. I : Attr = Blk;

10. C > E : Attr = Blk

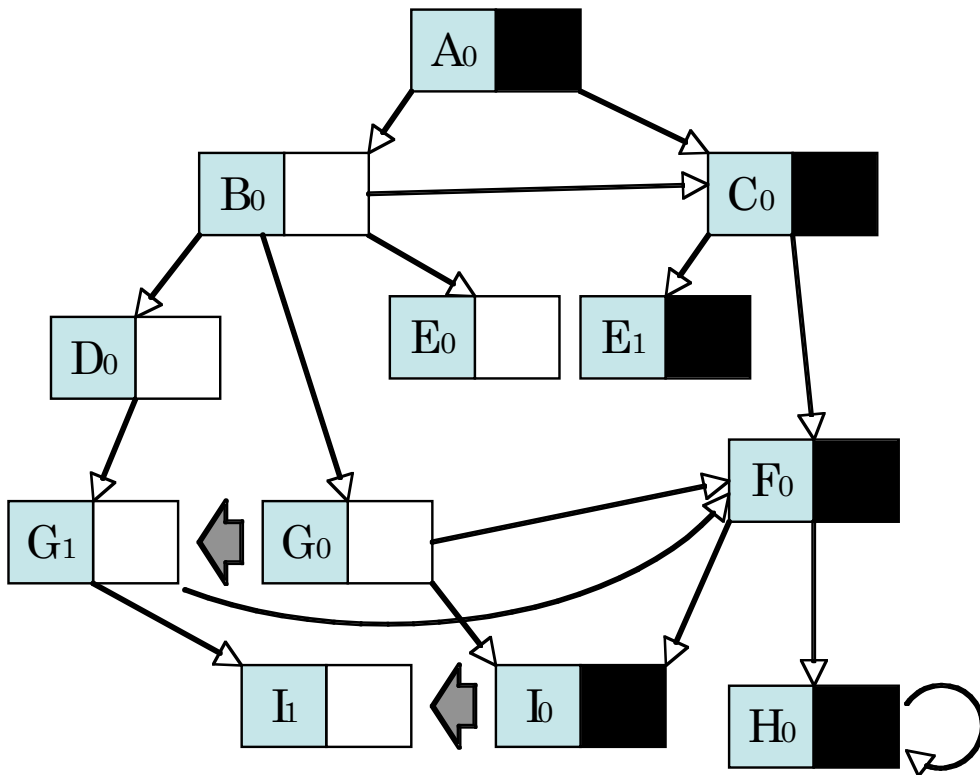
Adding Rule 10



10. C > E : Attr = Blk;

11. D > G > I : Attr = Wht

Adding Rule 11



11. D > G > I : Attr = Wht;

Outline

- Introduction
- Virtual Programming Environment
- Specification Rules
- Reconfiguring Distribution
- **Related Work**
- Discussion and Conclusion

Distributed Programming

- Transparent distributed computing in JAVA:
 - JavaParty, Addistant, ProActive, Actor Foundry, SALSA
 - Special “relocatable” objects
 - Lack of structured deployment schemes
 - Little support for customizing loading timing and order
- Web toolkits and frameworks
 - Better abstraction for Web programming
 - Prototype.js / Dojo : JavaScript
 - Echo2 / Google Web Toolkit : Java
 - XML II / XMLVM : display protocol and special JVM

Metaprogramming

- Programming on programs
- Meta Object Protocol (MOP)
 - Meta objects control base objects
 - Requires runtime support
- Aspect-oriented Programming
 - Programming on aspects (concerns)
 - Weaver: add concerns back
- Powerful, flexible but not easy to use

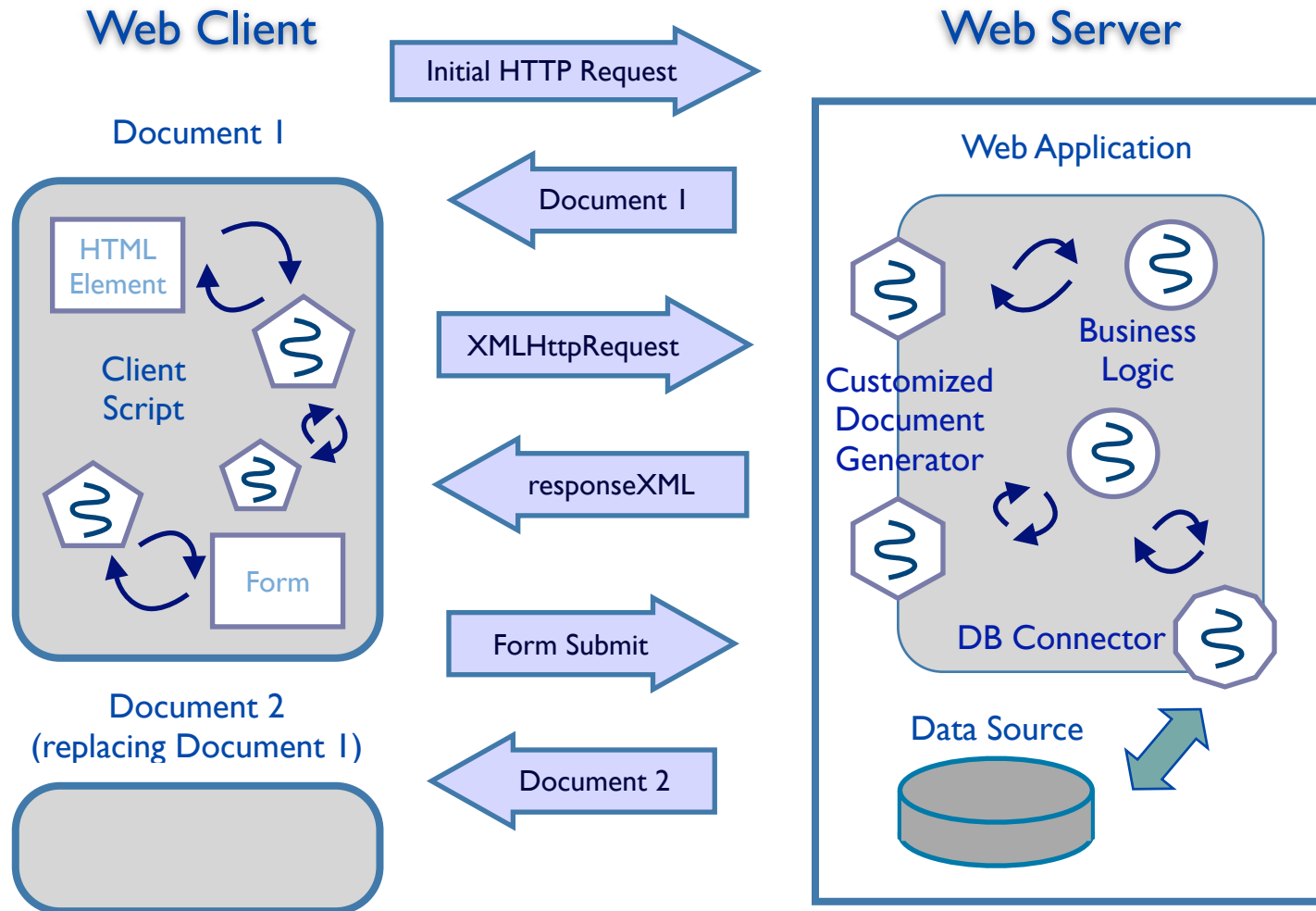
Outline

- Introduction
- Virtual Framework
- Specification Rules
- Reconfiguring Distribution
- Related Work
- **Discussion and Conclusion**

Discussion and Conclusion

- Customizable Web applications through
 - Virtual framework
 - Specification system
- Actor model is extensible
- ActorSpec is flexible
- Ongoing work (DAIS 2007)
 - Modular specification language
 - Context-aware Web applications

Anatomy



ActorSpec

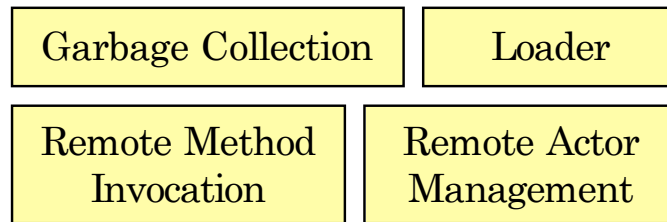
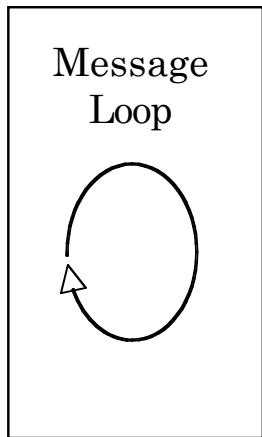
- Specification System
 - Express concerns available in the runtime environment
 - Connect concerns to application logic
- Decoupled from programming and runtime environments
 - Runtime environment: protocol for concerns
 - Application: prototype names and composition structure

Specification on Prototypes

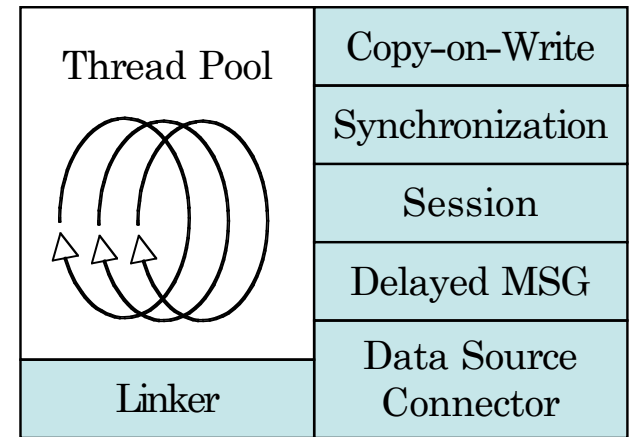
- Prototypes are also components
 - Need specification, too. e.g. loading policy of code
- Specification rules also apply on prototypes
 - [actor feature] : attribute = value
 - Those actors require a special prototype implementation
 - PricePanel > GridControl : PrototypeLoad = PreLoad
 - CalcPanel > GridControl : PrototypeLoad = OnDeman

Implementation

Client Podium



Server Podium



JavaScript
Library

Tomcat 5.5
Rhino 1.5
Java Library

Attributes of Distribution

- Location for actors and prototypes:
 - *Client*: created and works in client
 - *Server*: created and works in server
 - *Mobile*: created in server but works in client
- Loading policy for mobile actors
 - *PreLoad*: loaded when its reference exported
 - *OnDemand*: loaded when the first message received
- Loading policy for client prototypes
 - *PreLoad*: loaded when any of its referrer loaded
 - *OnDemand*: loaded when the first creation request

Proof

- The algorithm stops (in polynomial time)
- Generated ILG is correct implementation of PDG
 - A path in PDG has a unique matching path in ILG
- ILG follows the specification rules
 - Vertex's value follows the latest added rule