Askemos - Holons at the Market Place

Jörg F. Wittenberger

softeyes

Joerg.Wittenberger@pobox.com

www.askemos.org

The Global Computer is not an object
it is a property.

Ben Howell Davis

fair, unbreakable, fraud proof

Askemos

• advanced infrastructure •

• redundant distributed execution environment •

• Single, Abstract Virtual Machine •

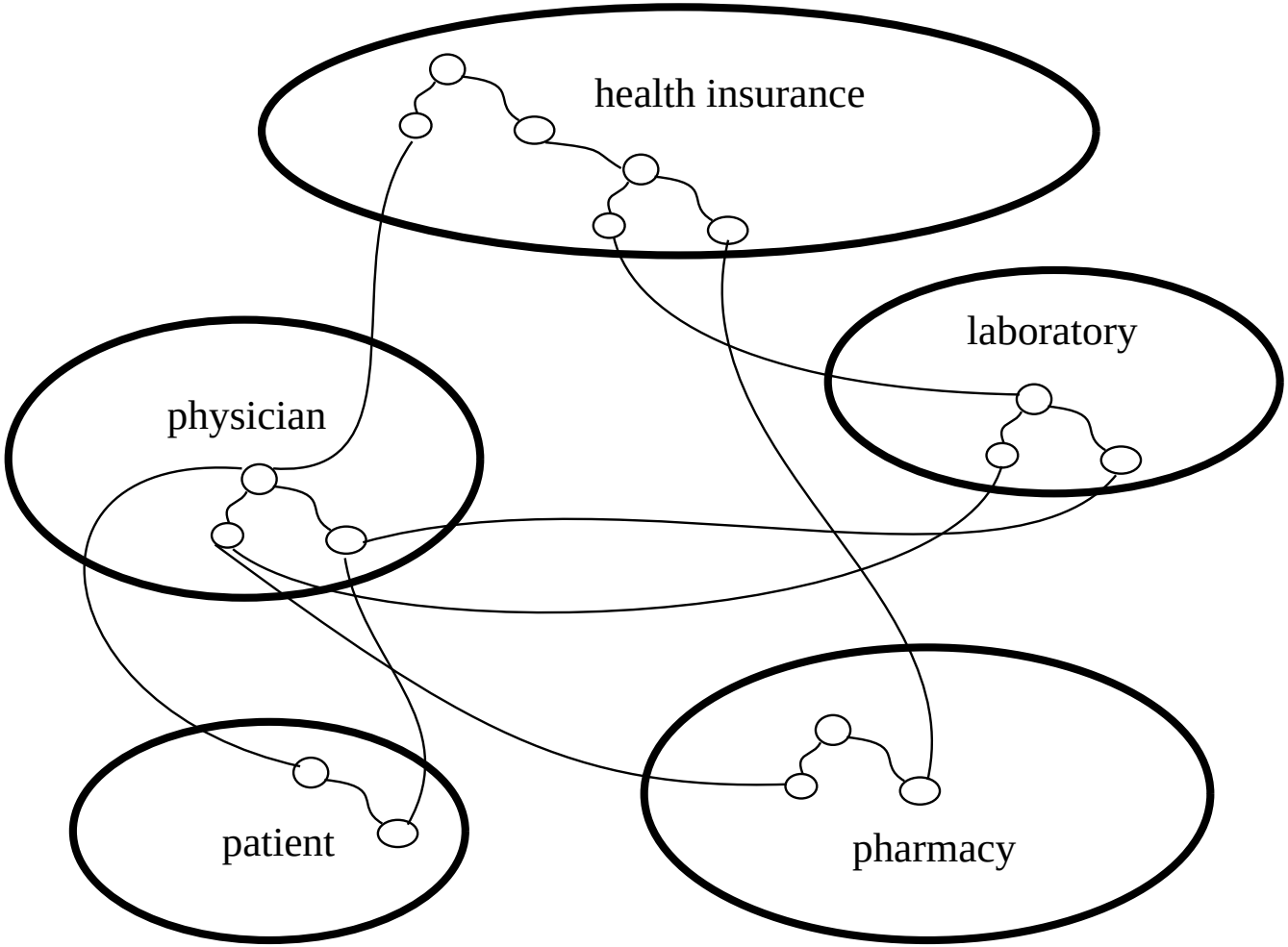
• System API in terms of XML documents •

• failure and intrusion resistant •

Agents at the Market Place

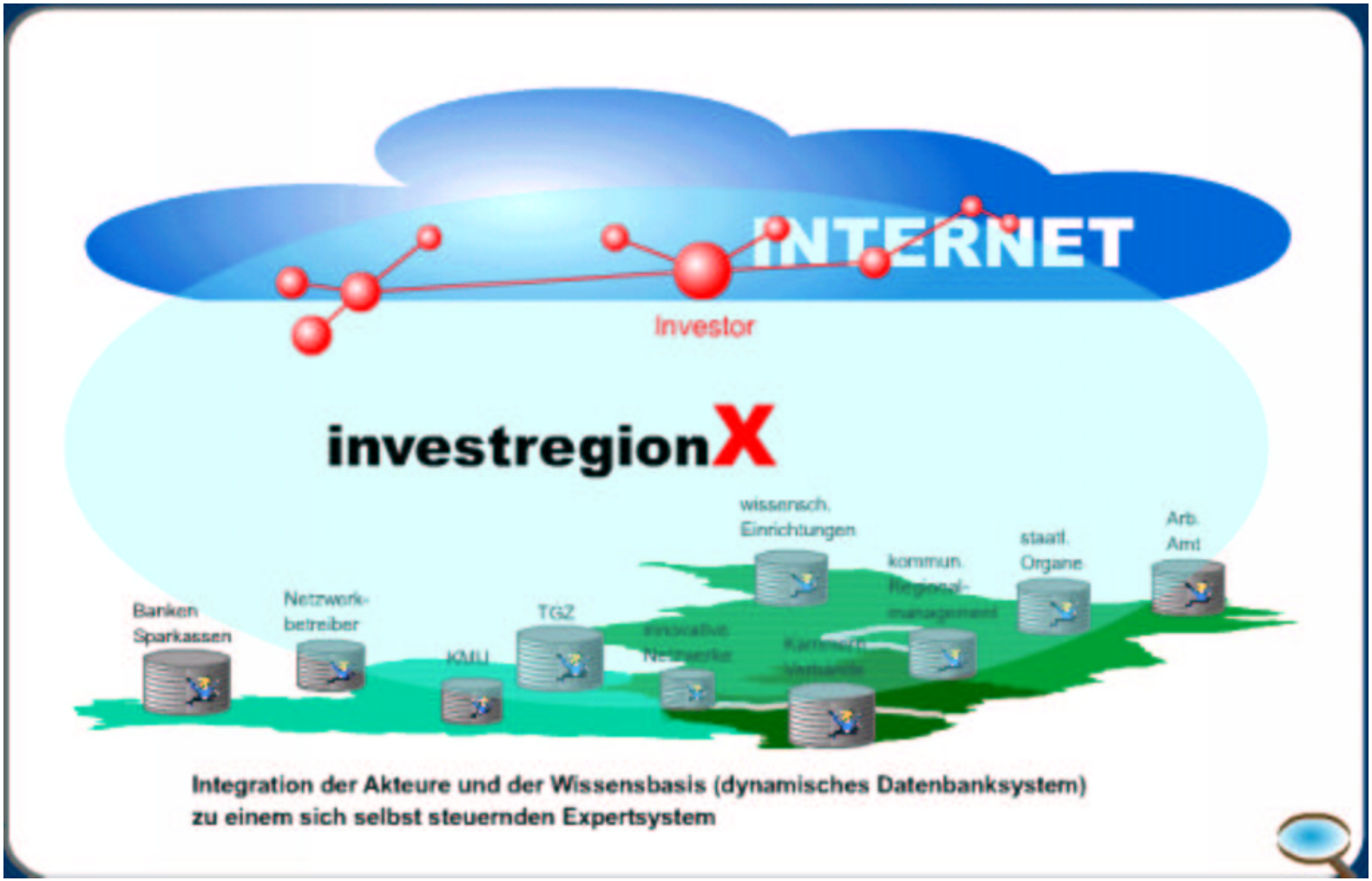
- Why do we need it?
- Requirement Analysis
- The Implementation
- How to use?
- Comparison
- Further work

Motivation



Interactions and relationships in health care.

Motivation



Collaborative commerce in rural development.
www.investregionx.de

The Problem

Holons might represent a legal body;

Some of those are **not** social, MAS mismatch here because of agents intrinsic properties:

- **veracity:** an agent will not knowingly communicate false information
- **benevolence:** agents do not have conflicting goals, and that every agent will therefore always try to do what is asked of it;
- **rationality:** agent will act in order to achieve its goals, and will not act in such a way as to prevent its goals being achieved – at least insofar as its beliefs permit;

Requirements

The enabling infrastructure for e-Business, e-Medicine and e-Government - how would it be?

- stores fraud and privacy sensitive data
- needs access control for intellectual property
- needs (bearer) certificates, transactions, communication
- is based on mutual trust between partners,
  no central authority deemed acceptable
- requires reliability, liability, availability
- needs scalability, efficiency, flexibility and standards

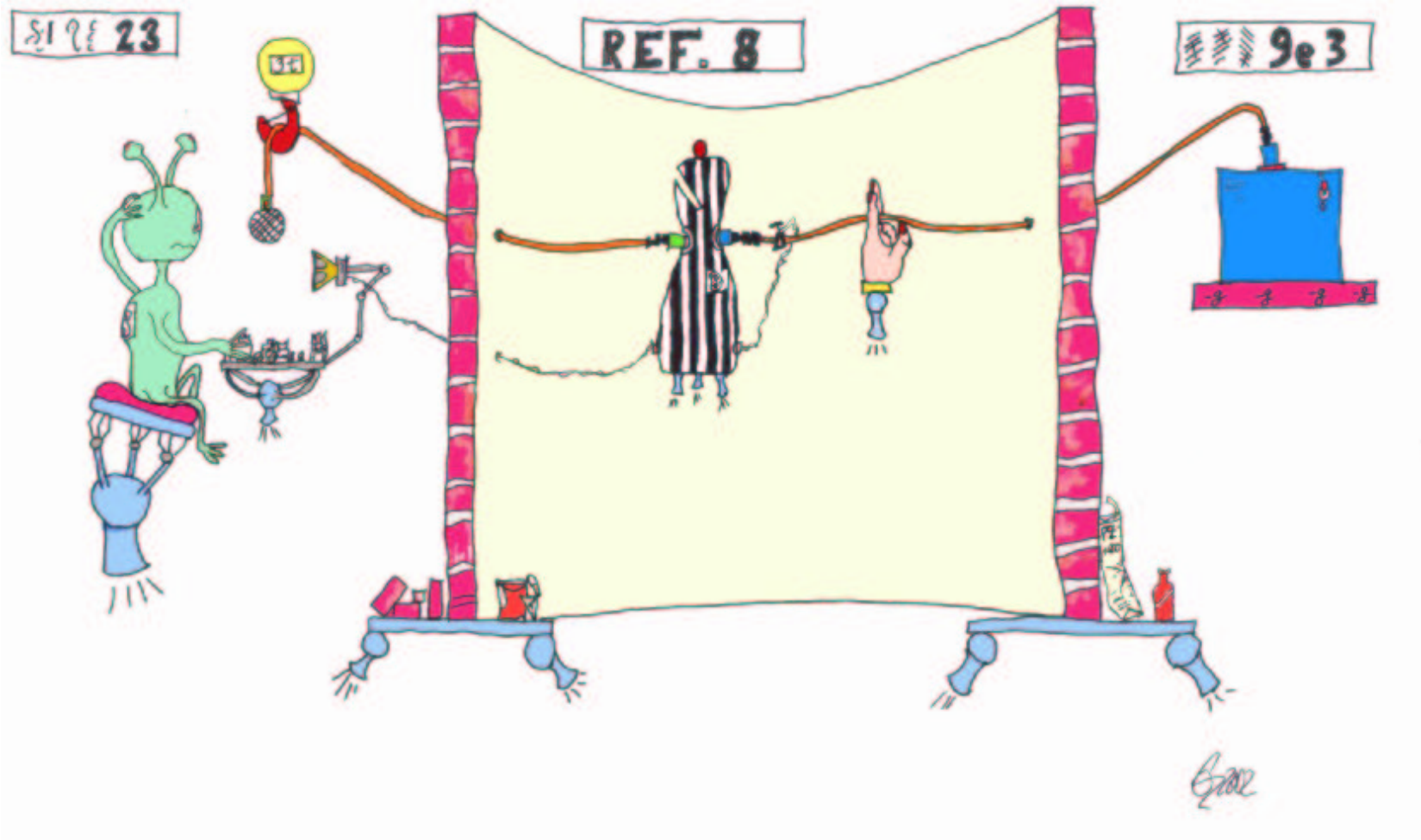
Challenges

- parallel and independent development
- rapid changing work flow
- smooth transitions between versions
- unstoppable processes (frozen)
- physical equipment just breaks $\Rightarrow$ data in danger
- information is not physical, why should it break?
- need to make small equipment less sensitive
- build reliable system from unreliable components

A “Legal” Abstract Machine

- formal semantics in order to be
  - fully understood
  - decidable in case of dispute
- fraud proofed, at least traceable
- fair, democratic, decentralized

Natural Pattern



Artwork by Christoph Zurnieden

Implementation 1: Something Is Somewhere

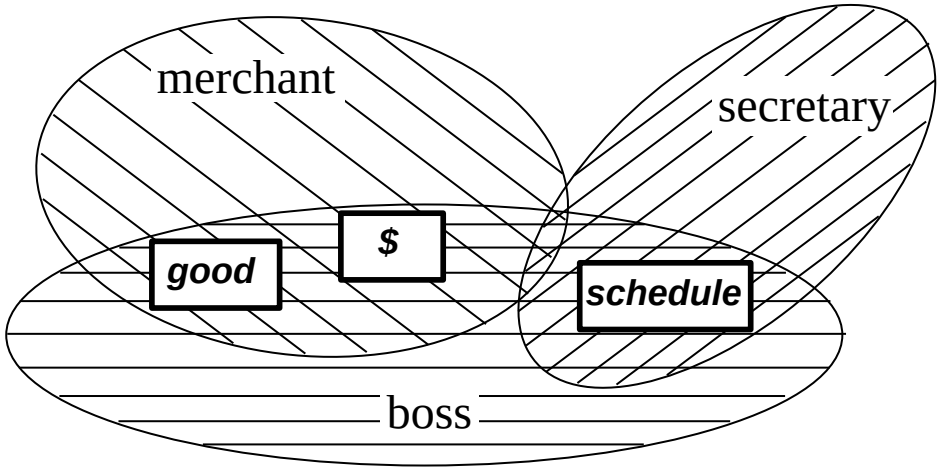
the concept of the information space

- main concept: legal “space” connecting legal bodies
- information is a singleton, data is not

$$\frac{information}{data\ representation} = \frac{1}{many}$$

- copying is evil (money, acknowledgement of intellectual property)
- majority counts - byzantine agreement
- new, emergent abstraction: truth

Implementation 2: Location Is Protection



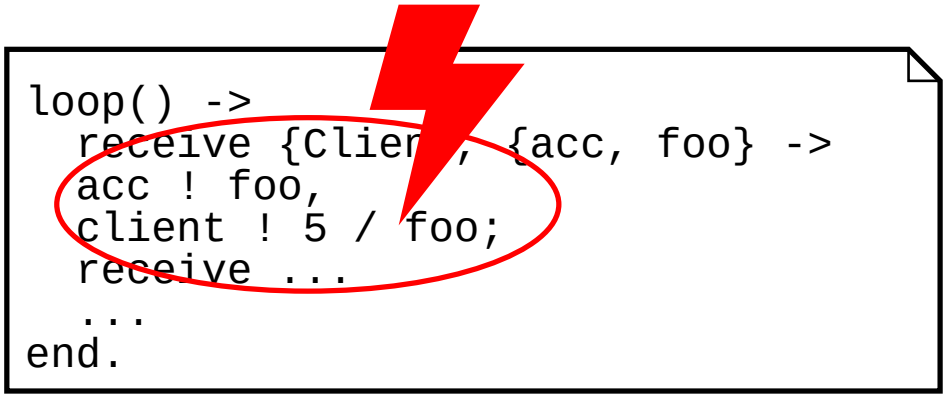
- overlapping realms of rights
- no central authority to abuse
- formal model: set theory

Implementation 3: Static Assertions

- the cost of assignment; SICP we know there is no theory for assignment
- recently proposals on formal semantics for imperative languages: horribly complicated
- functional programming found ideal for static aspects
- functional programming is well understood and easy to teach

Implementation 4: Dynamics

Example: an Erlang message loop:



```
loop() ->
  receive {Client, {acc, foo} ->
  acc ! foo,
  client ! 5 / foo;
  receive ...
  ...
end.
```

nondeterminism: division by zero, message to ACC is already sent, CLIENT might get a message

- sequential execution in loops introduces strange dynamic semantic
- hard to model
- way out: atomic transitions formalized by Petri Net

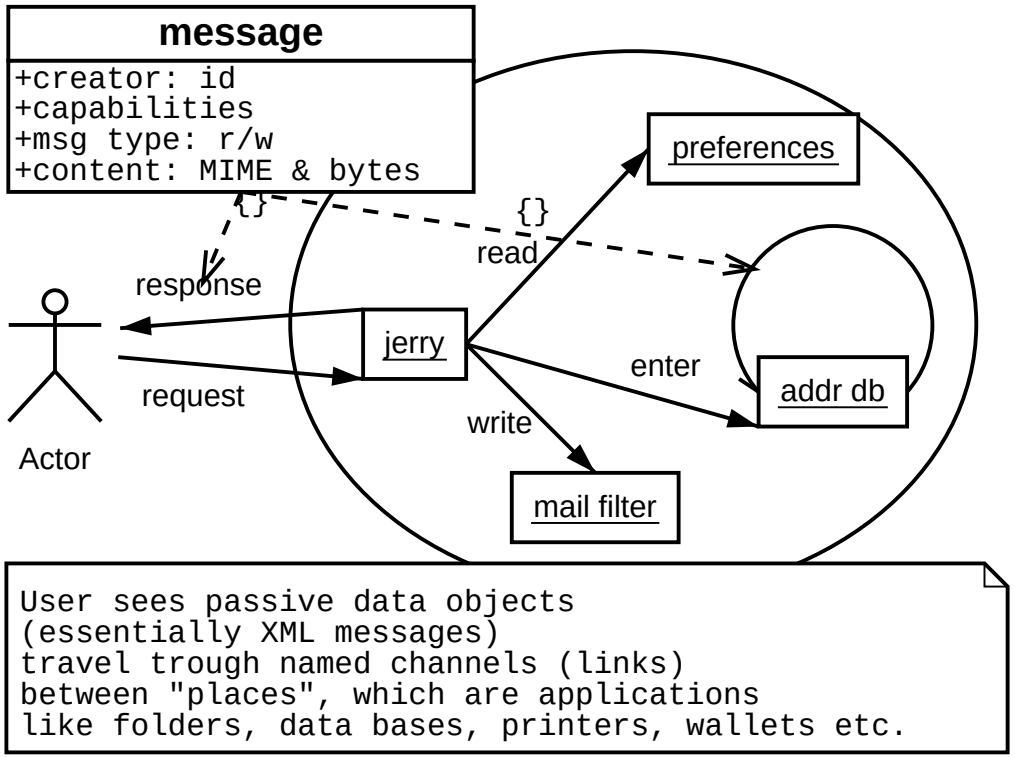
Implementation 4: Dynamic Behavior

```
stylesheet>
  . . .
 template> match = *[@type='write']
  reply>
   send> to> copy-of> select = $message/acc
         Body> copy-of> select = $message/foo
   send> to> copy-of> select = $message/client
         Body> copy-of> select = 5 / $message/foo
```

Notation: SOX, "human editable" XML
full AST in XML; not practical; I want a different syntax (e.g. DSSSL, Haskell)

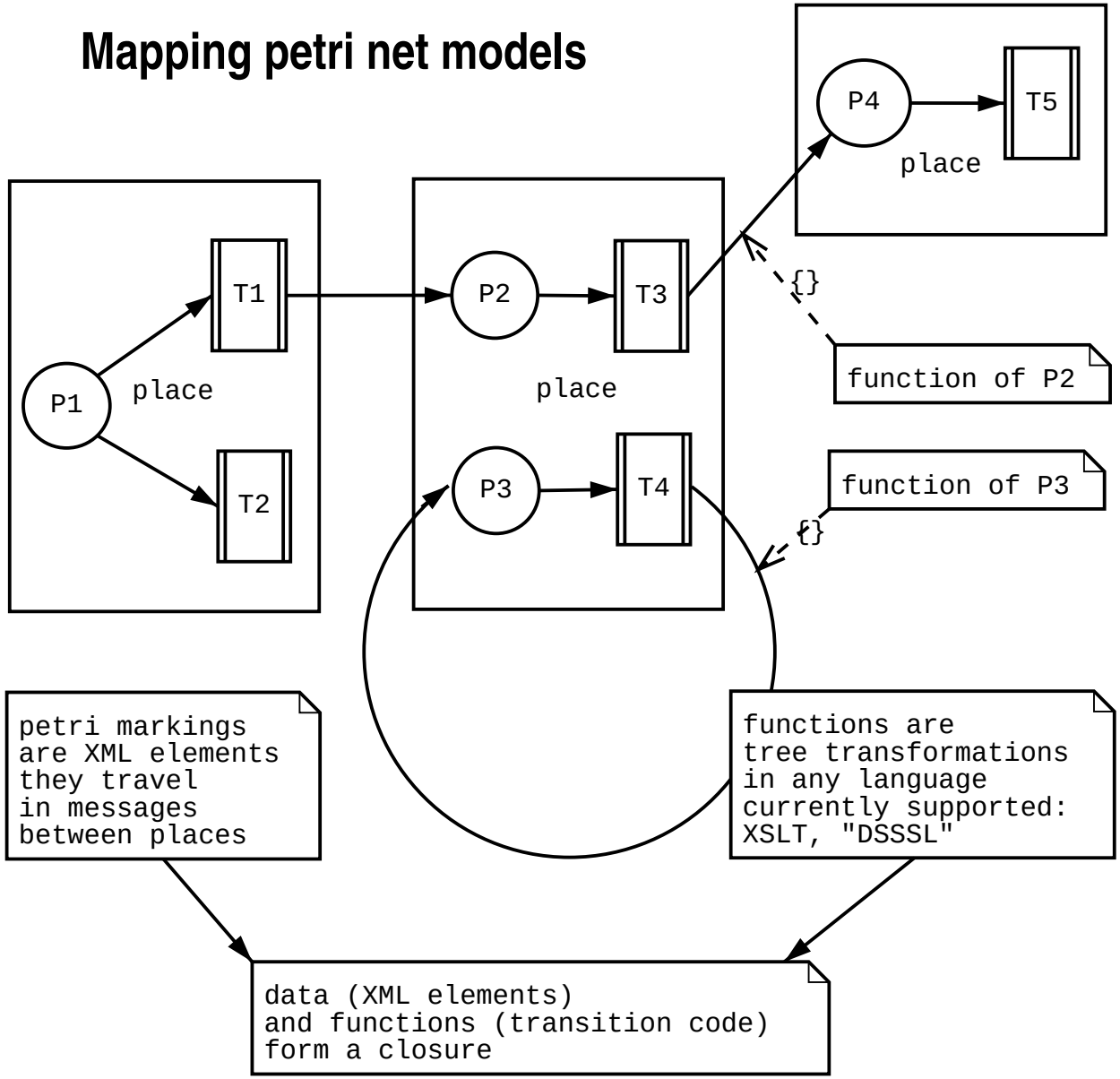
- implicit, message driven control flow
- template applied when message arrives
- atomic transition, division by zero -> no message at all
- can check all messages for security violations

User Level View

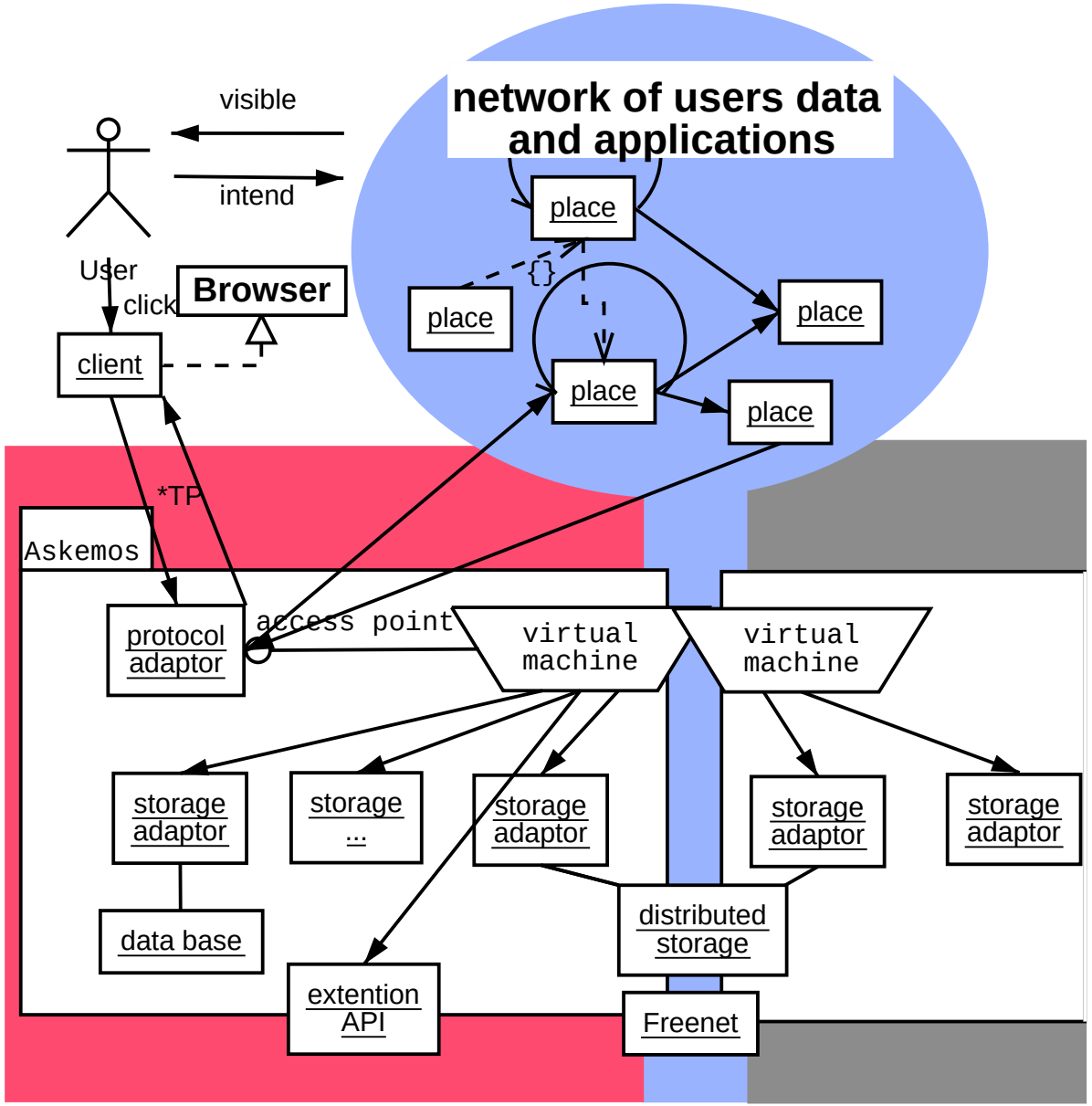


- autonomous objects or agents at places in the information space
- those function blocks communicate via asynchronous messages

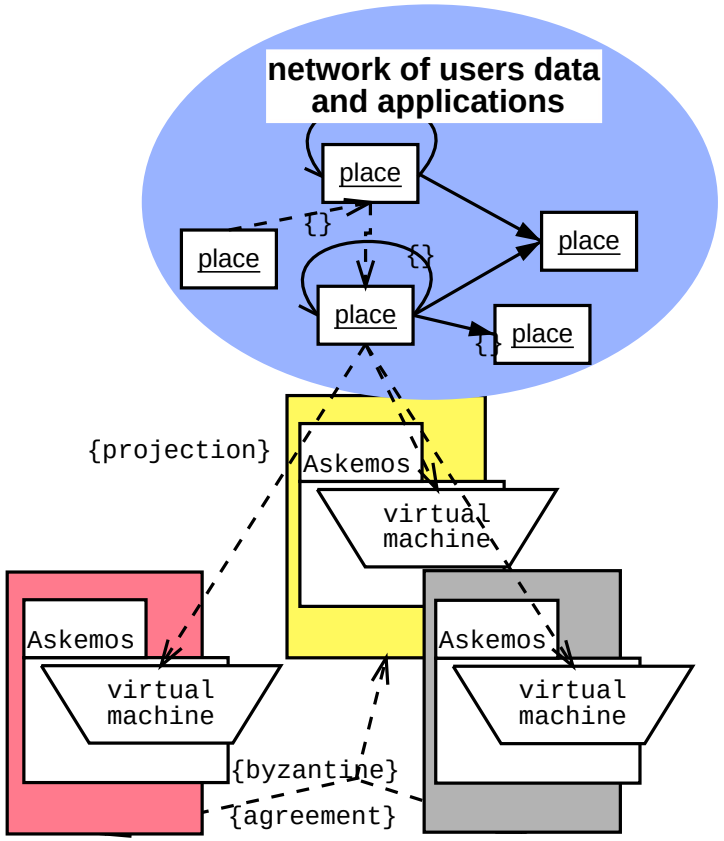
Formal Model



Architecture

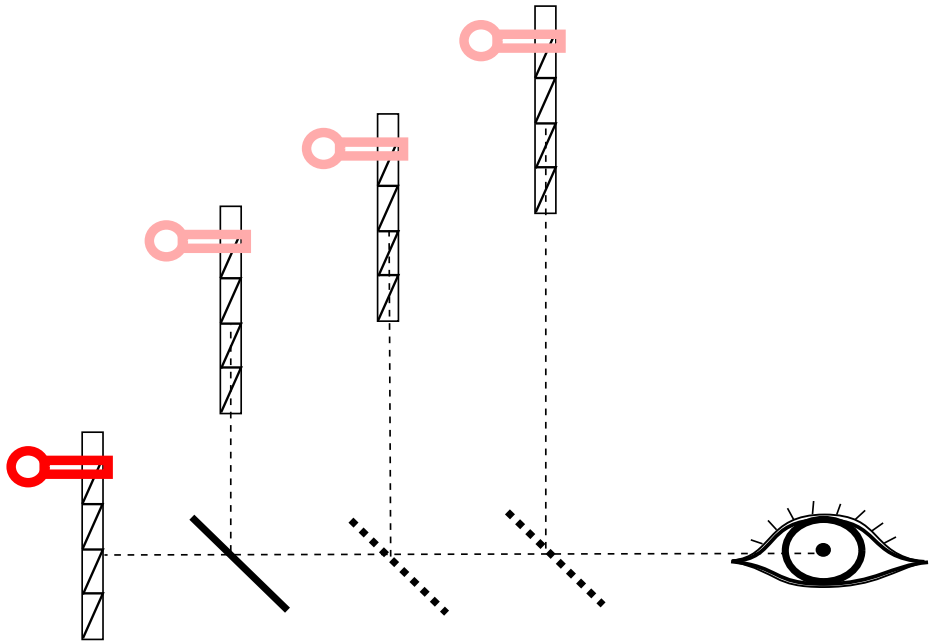


Projections



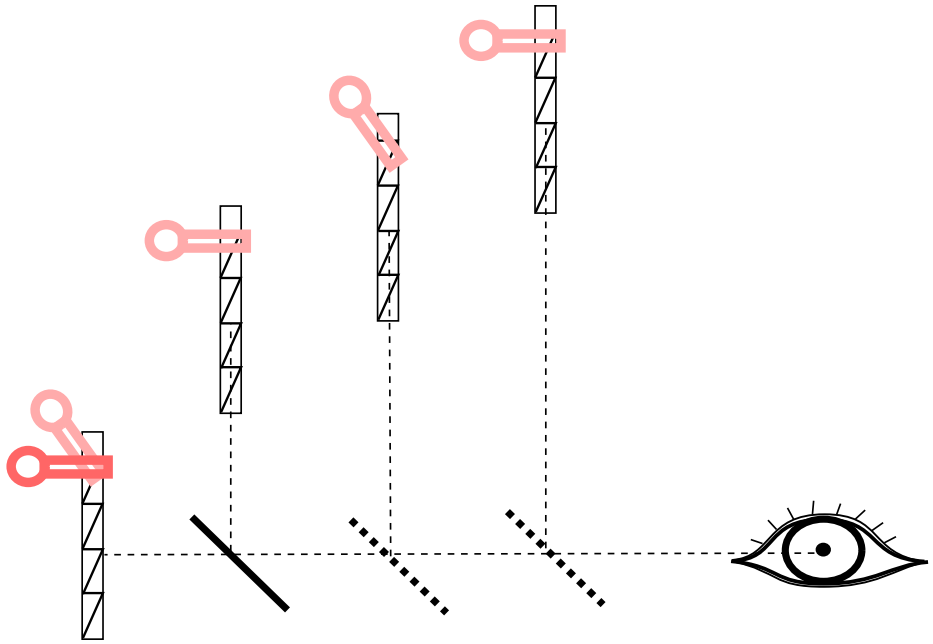
- each place projected into the belief of several agents which obey contract of proper execution
- mutual verification (byzantine agreement) $\Rightarrow$ trust

Projections



- each place projected into the belief of several agents which obey contract of proper execution
- mutual verification (byzantine agreement) $\Rightarrow$ trust

Projections



- faulty agents are spotted

Slave of Theory

- truth by mutual verification

  $\Rightarrow$ byzantine agreement

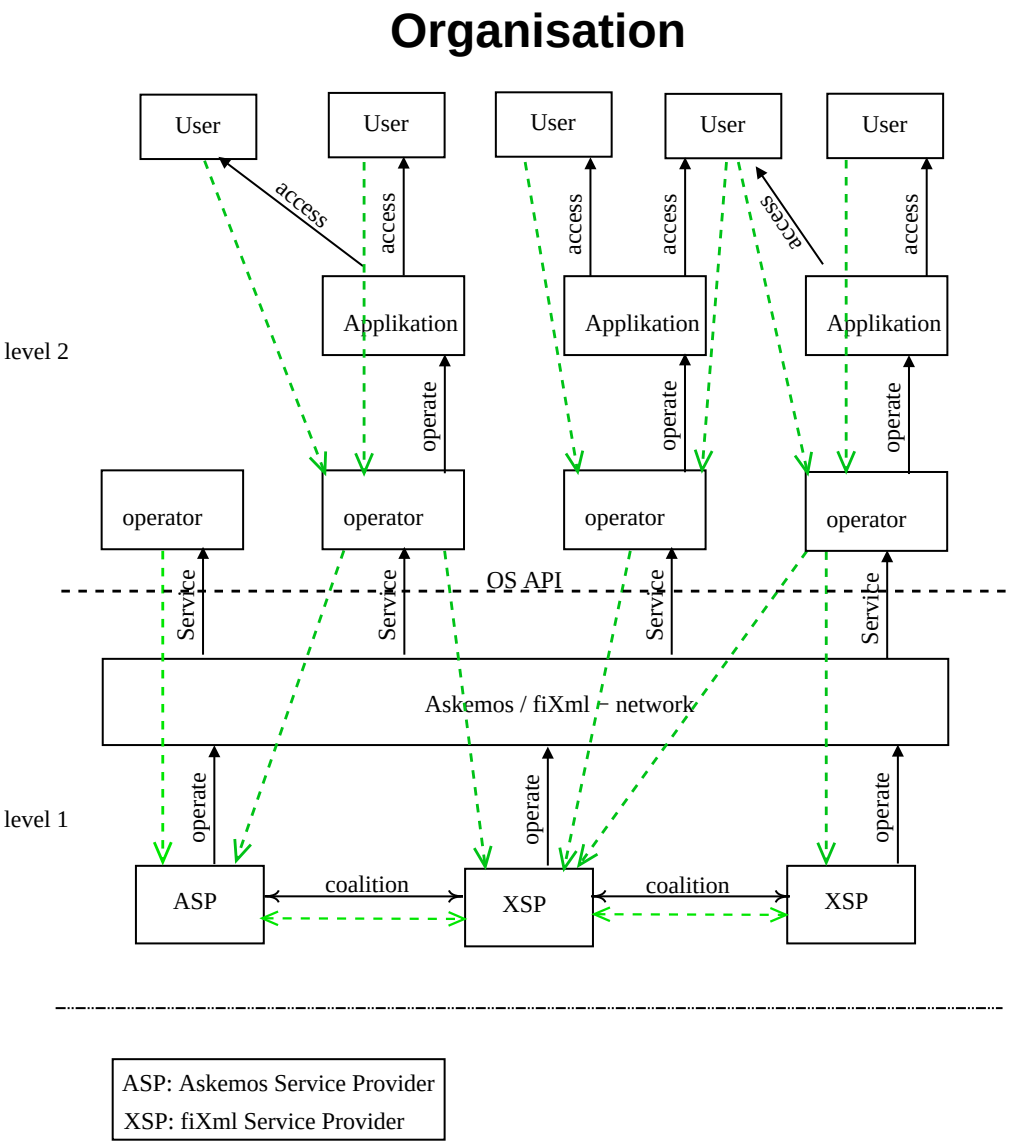
- cleanly defined semantics require sound theory

  $\Rightarrow$ set theory + functional programming + petri net

- different implementations may have different representations

  $\Rightarrow$ still agreement needed - core API in XML
  (standard is better than better)

computation out of the wall plug



Code Example: MVC frame

```
<xsl:stylesheet …>
 <xsl:template match="*[@type=&quot;read&quot;]">
   . . .
   <!-- viewer code -->
   . . .
 </xsl:template>

 <xsl:template match="*[@type=&quot;write&quot;]">
  . . .
  <!-- controller -->
  . . .
 </xsl:template>
</xsl:stylesheet>
```

Code Example: View Code

```
<xsl:template match="*[@type=&quot;read&quot;]">
 <html><head><title>One Cent</title></head>
   . . .
   <dsssl:if test="(service-level)">
    <p>You can grab this bill by pushing the button below.</p>
    <form method="post">
    . . .
   </dsssl:if>
 </html>
</xsl:template>
```

Code Example: Controller

```
<xsl:template match="*[@type=&quot;write&quot;]">
  <mind:reply>
   <dsssl:copy-of select="(document-element (grove-root (current-node)))"/>
    <dsssl:if test="(service-level)">
     <mind:protection>
      <dsssl:copy-of select="#CONTENT">
(let ((t (data (form-field 'offer (current-node)))))
 (if (equal? t "")
     (symbol-&gt;string (msg 'dc-creator))
     (string-append
      (symbol-&gt;string (msg 'dc-creator))
      "/" (oid-&gt;string (entry-name-&gt;oid t)))))
      </dsssl:copy-of>
     </mind:protection>
  </mind:reply>
</xsl:template>
```

Comparison

FramerD

nice language, side effects, moving target

Erlang

obvious similarities, either change it or redo (overhead)

Java

is a platform: many libs, Java VM has magic functions, hardly secure, battle field

FIPA

huge, now incomplete standardization effort; not everything needed

reality check: XML+HTTP is the only widely deployed standard

Further Work

- deploy in more pilot projects
- complete implementation, port to other platforms
- check for possible FIPA compliance
- provide features from similar environments

Conclusion

- trustworthy, distributed virtual execution environment
- suited for modeling holarchies comprised of legal bodies
- might become almost FIPA compliant but difference concerning mobility will stand

Askemos - Holons at the Market Place

Thank You!

Joerg.Wittenberger@pobox.com

www.askemos.org

Relevant Standards

XML	(W3C)	mandatory; data (tree) representation
XSLT	(W3C)	mandatory; tree transformation
SOAP	(W3C)	mandatory; object access
HTTP	(IETF)	mandatory; network transport
DSSSL	(ISO)	optional; tree transformation
61499	(IEC)	optional; function blocks
FIPA	(FIPA)	optional; agents technology

Deeds - Content Defined Identifiers

```
(define deed-slots
  '(dc-creator      ; creator (Dublin Core)
    dc-date         ; creation date
    action-document ; behavior
    body            ; serialized content
    ))
```

- object **identifier** is cryptographic check sum
- name ≠ identifier:
  - names are intentionally chosen, human readable, reusable, might have structure
  - identifiers are intrinsic, one-to-one non-ambiguous
- deeds don't change - verification easy

How To Identify Dilemmas?

iteration
not first-time out
participatory design

dialogue

implementor ⟷ **users**

bug reports
code
⋮

Gregor Kiczales (1994)