

Programmable Messaging for Electronic Government

Building A Foundation

Elsa Estevez and Tomasz Janowski

Center for Electronic Governance at UNU-IIST



UNITED NATIONS
UNIVERSITY

UNU-IIST

International Institute for
Software Technology

Overview

1	Introduction - Electronic Government	
2	Example - Electronic Licensing Service	
3	Programmable Messaging (G-EEG)	
	3.1	Concepts
	3.2	Behavior
	3.3	Model
	3.4	Application
	3.5	Implementation
4	Conclusions	

e-Government

Definition	Electronic Government refers to the use of ICT, particularly the Internet, as a tool to achieve better government. <i>[OECD]</i>
Aim	<ul style="list-style-type: none">● provide higher quality of public services● improve efficiency in government processes● improve communication with citizens, etc.
Drivers	<ul style="list-style-type: none">● processes – reengineering administrative processes● technology – integrated solutions, adoption of standards● people – technical/managerial training of public workforce

e-Government Maturity Stages

e-Government Maturity Stages:

1)	Emerging	static information on agency websites
2)	Enhanced	content updates and database searches
3)	Interactive	two-ways interactions with citizens
4)	Transactional	complete and secure on-line transactions
5)	Seamless	seamless services across agency boundaries

[UNPAN – UN Public Administration Network]

Stage 5 – Seamless Services

Seamless Service - Process

- a citizen specifies the need
 - a service is provided to fulfill the need
 - several agencies are involved in service delivery
 - a citizen is not aware of the government structure involved
-

Seamless Service - Challenges

- supported by cross-agency business process
 - regulated by policies that change over time
 - executed by different heterogeneous applications
 - delivered in a correct and reliable way
-

Example: Licensing Service

Issue a license for establishing a food and beverage business:

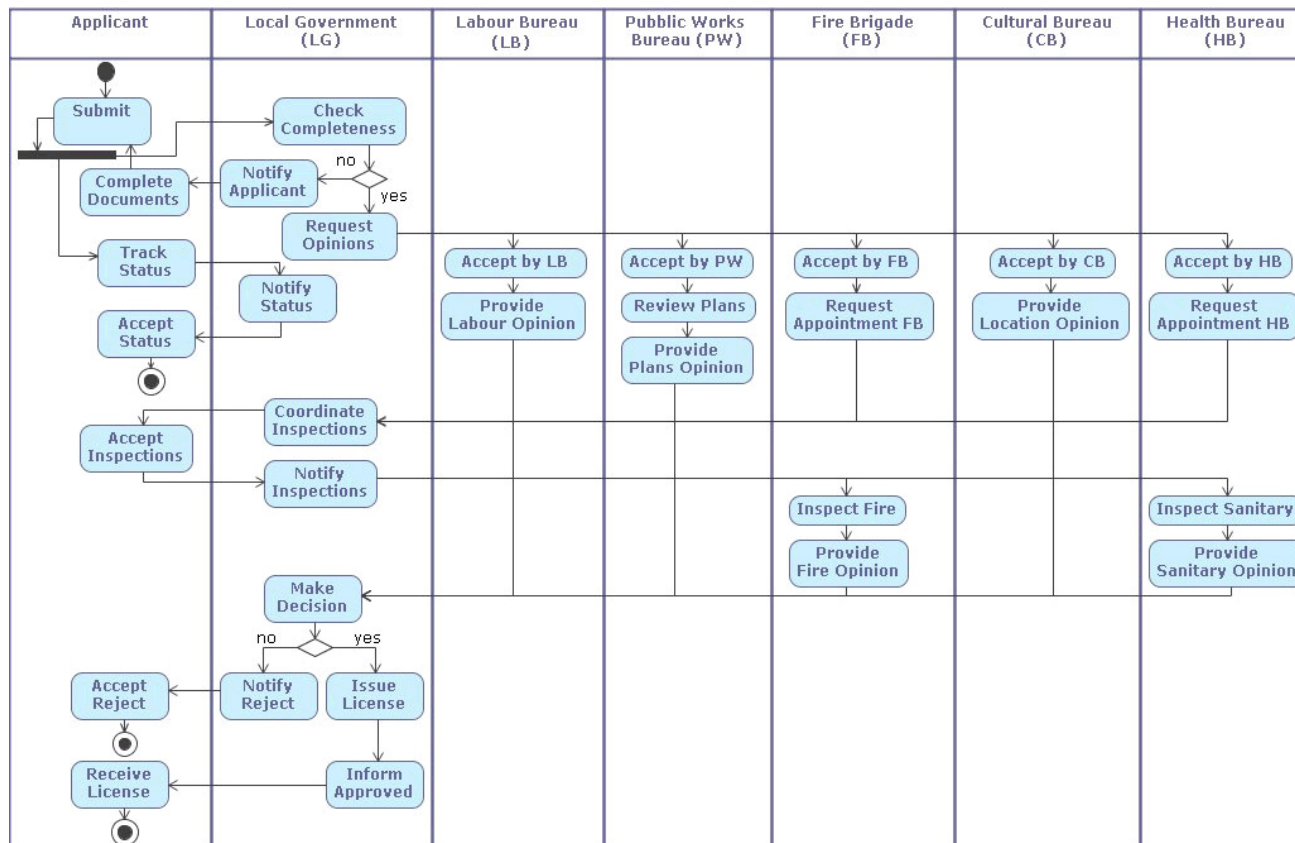
- 1) Submission - submitting application forms and documents
 - 2) Completeness Assessment - checking completeness of application
 - 3) Evaluation - Requesting opinions from other agencies
 - 4) Decision-making - issue or reject the application?
 - 5) Follow-up - Notification, appeals, etc.
-

Precise guidelines:



[IACM, MSARG]

Example: Licensing Process

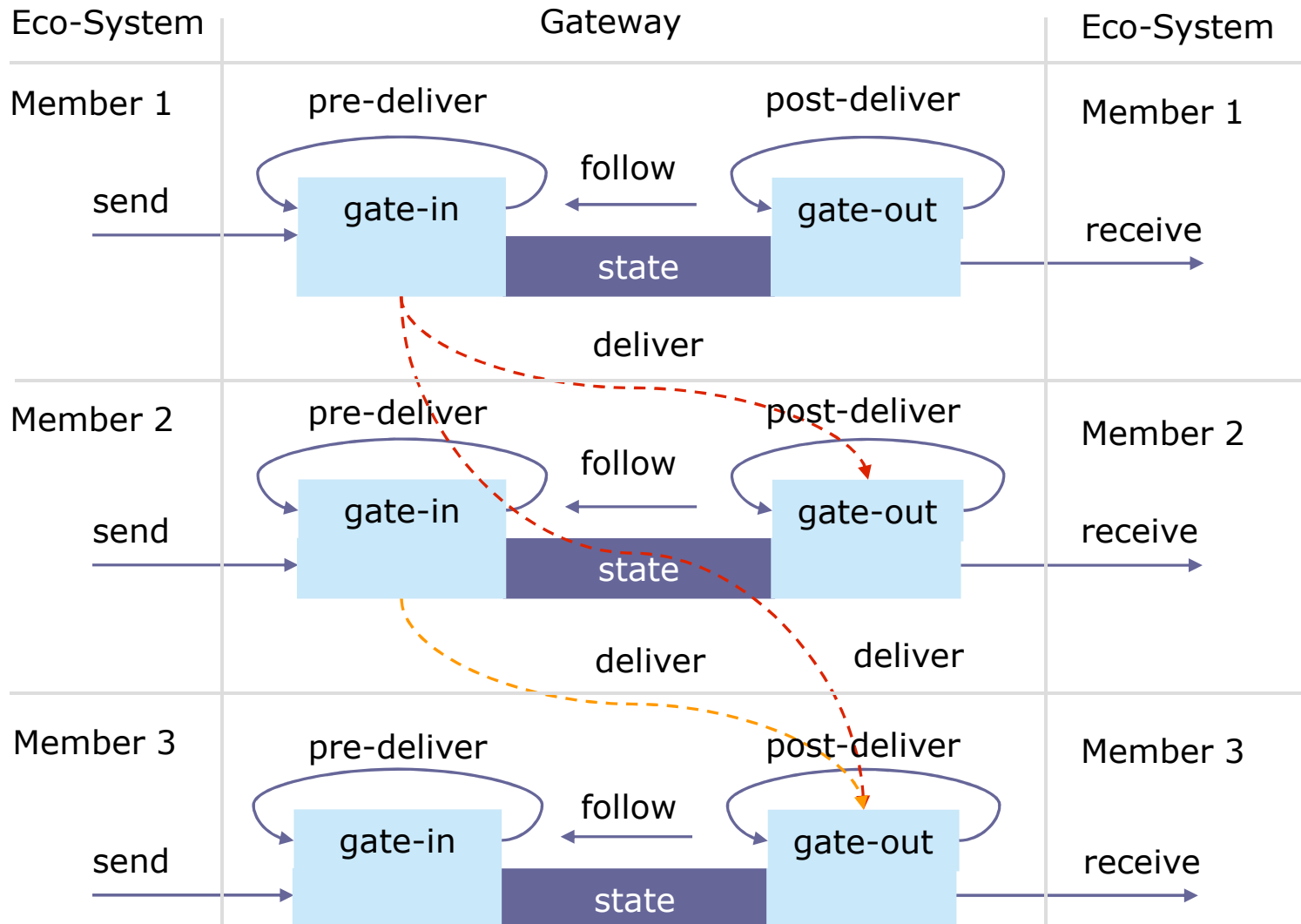


Aim: Develop middleware to support such processes – policy-driven exchange of messages between agencies, policies can change, rich messaging, etc.

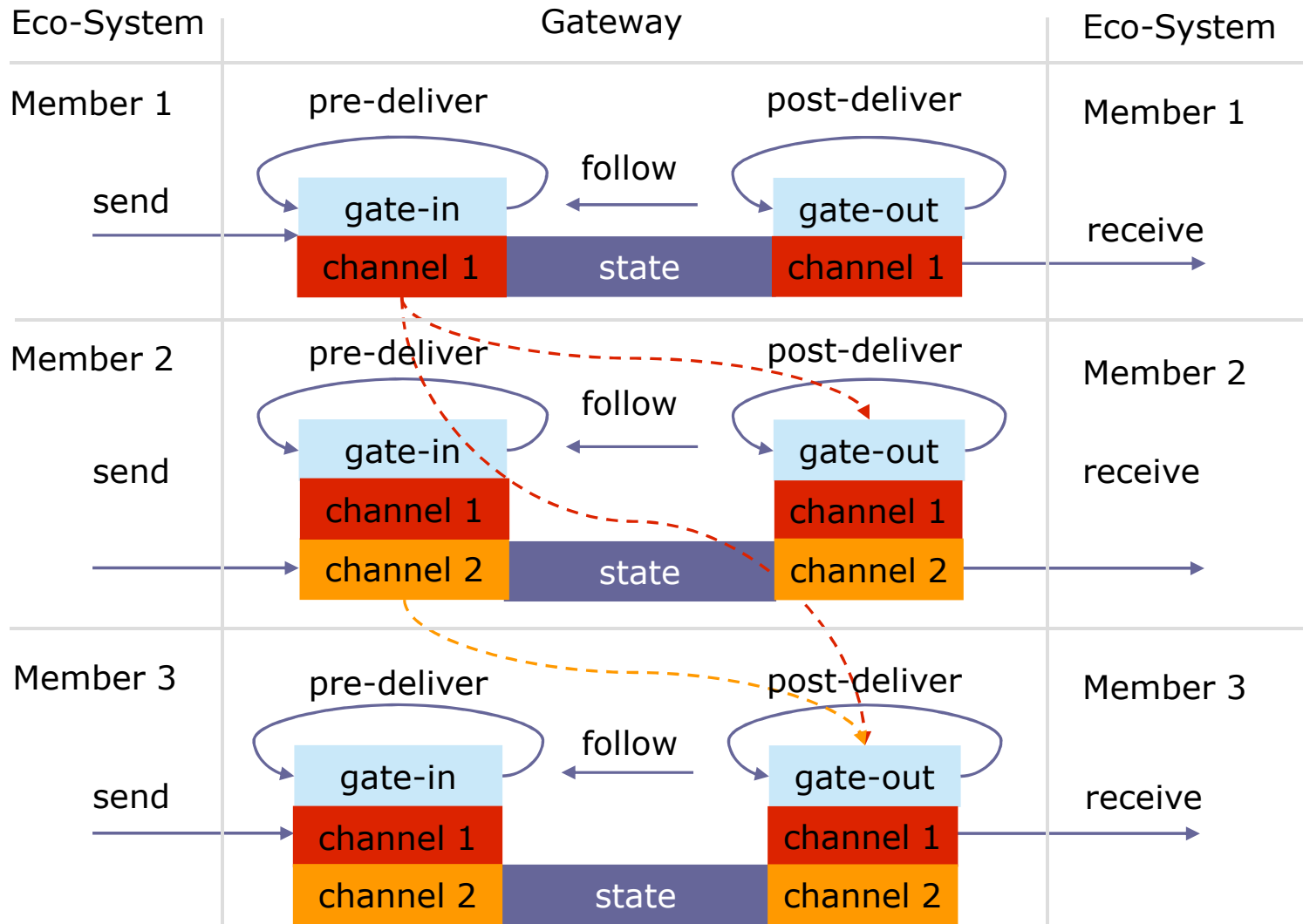
G-EEG

G-EEG	Government-Enterprise Ecosystem Gateway, a high-level communication and coordination framework for multi-organizational processes and applications.
G-EEG-CORE	A run-time framework allowing registered members to asynchronously exchange messages along dynamically created and subscribed channels.
G-EEG-EXTEND	A repository of process-independent and process-dependent extensions and a mechanism to dynamically enable them on top of G-EEG-CORE .
G-EEG-DEVELOP	A development framework to specify, design and verify messaging extensions, part of G-EEG-EXTEND: <ul style="list-style-type: none">- G-EEG-SPEC- G-EEG-DESIGN- G-EEG-VERIFY

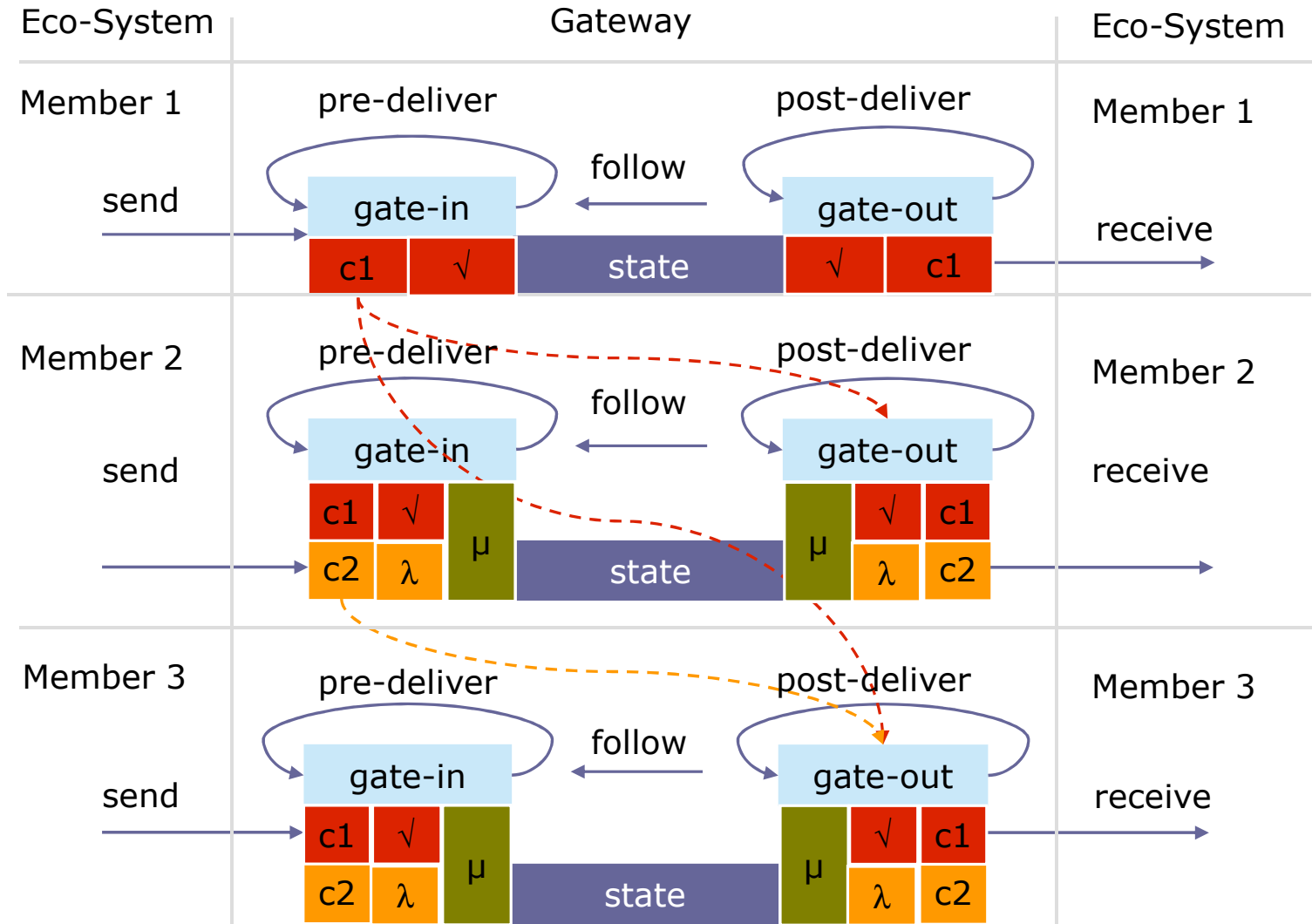
Concepts – High Level



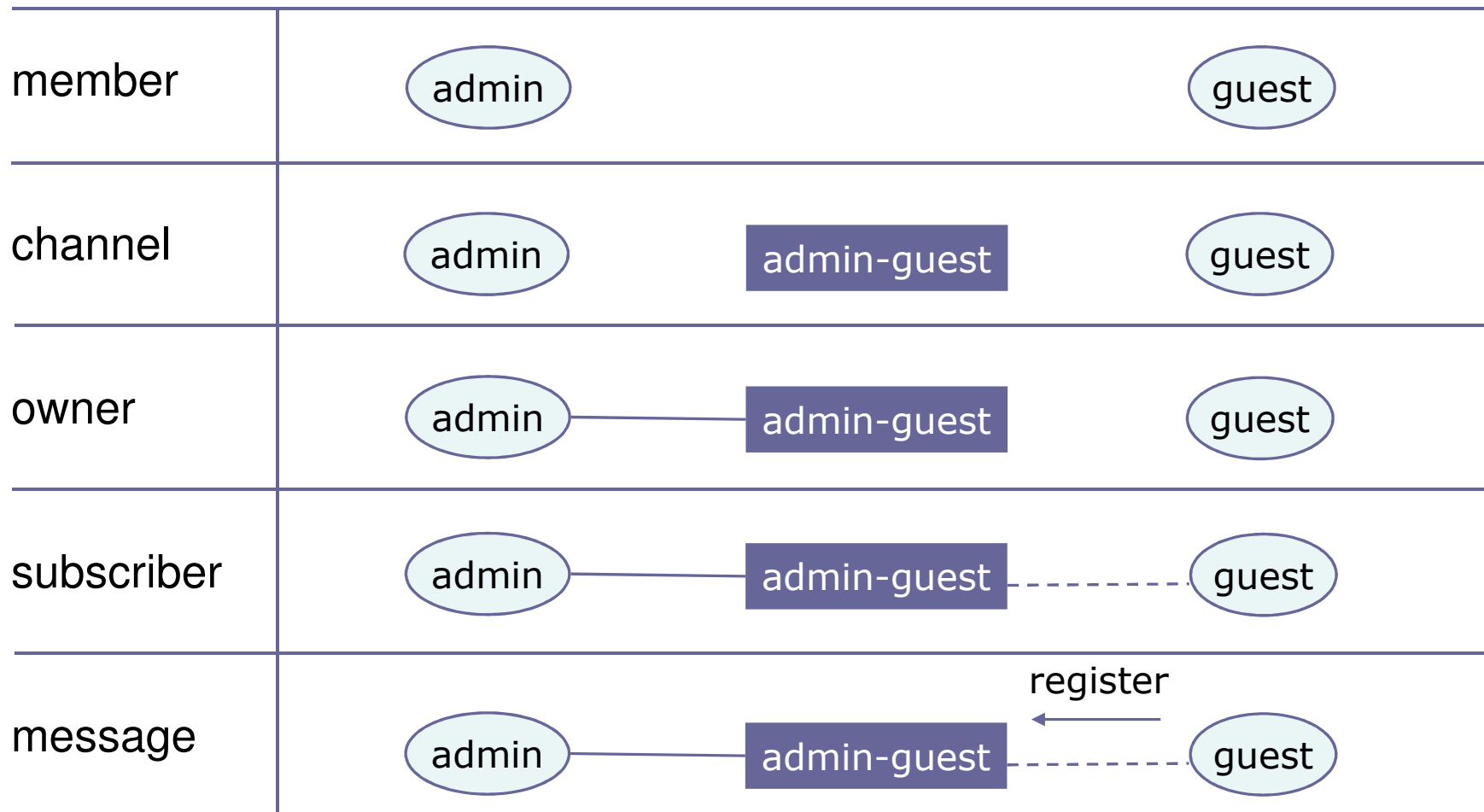
Concepts – Mid Level



Concepts – Low Level



Behavior - Graphical Notation

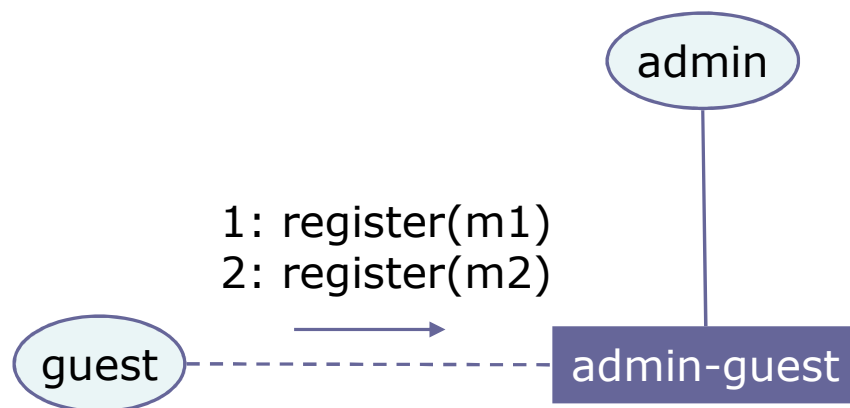


Behavior - Core Messaging

Core Messaging	1) register member
	2) create channel
	3) subscribe to a channel
	4) send and receive a message

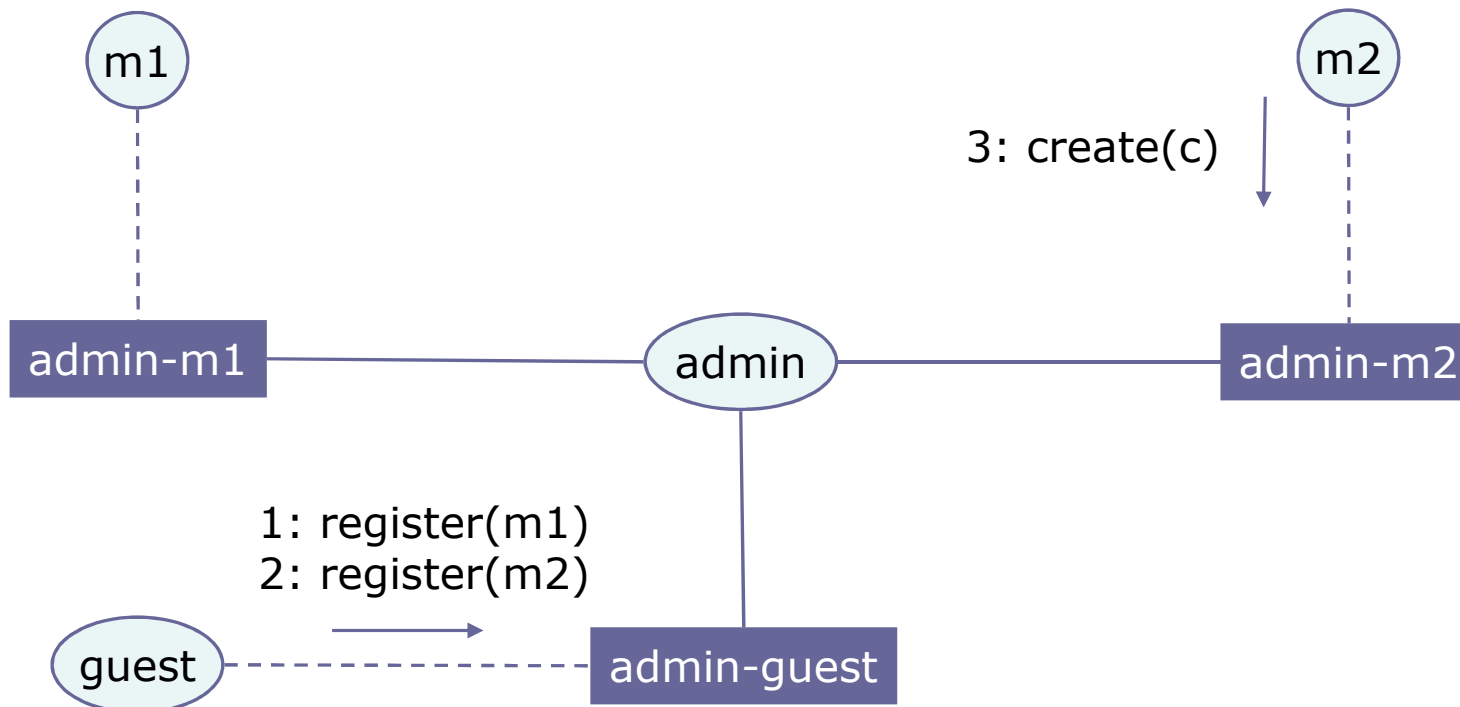
Behavior - Core Messaging 1

Register members:



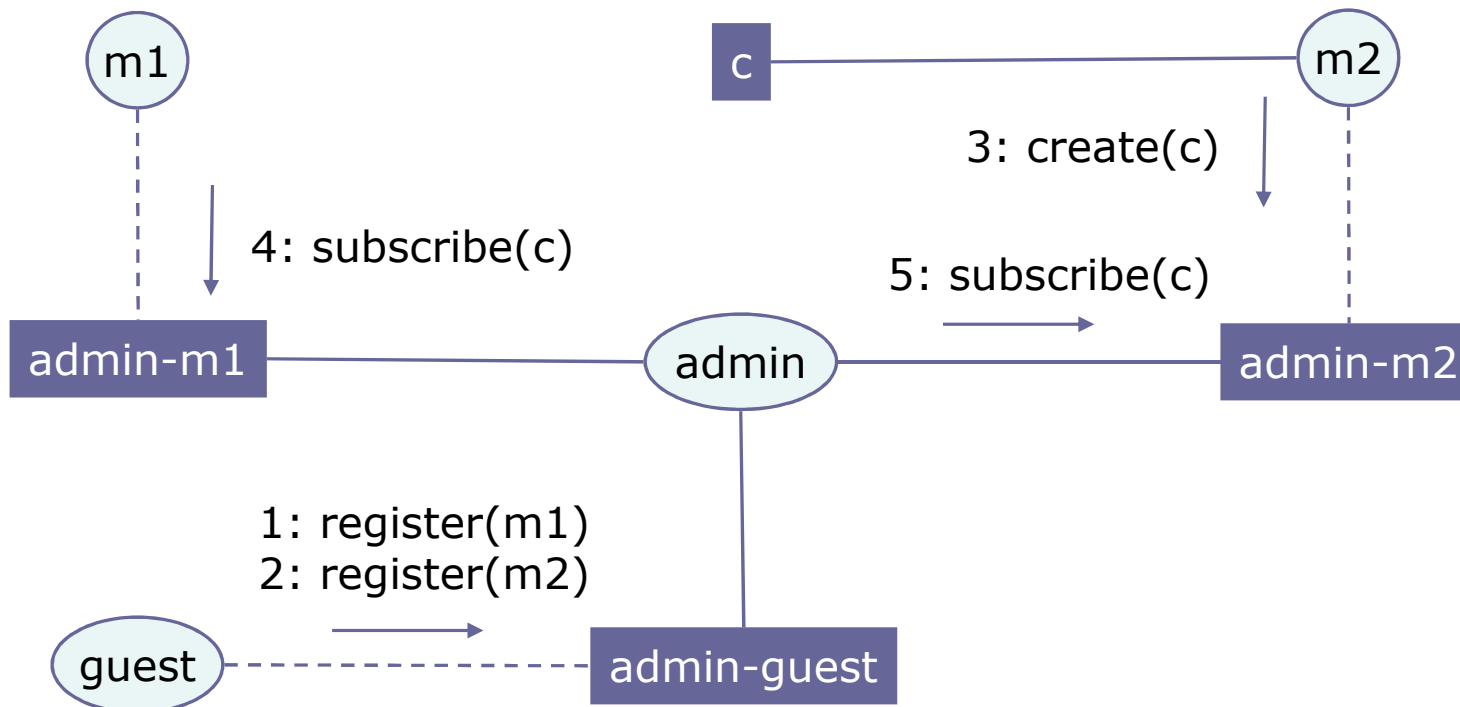
Behavior - Core Messaging 2

Create a channel:



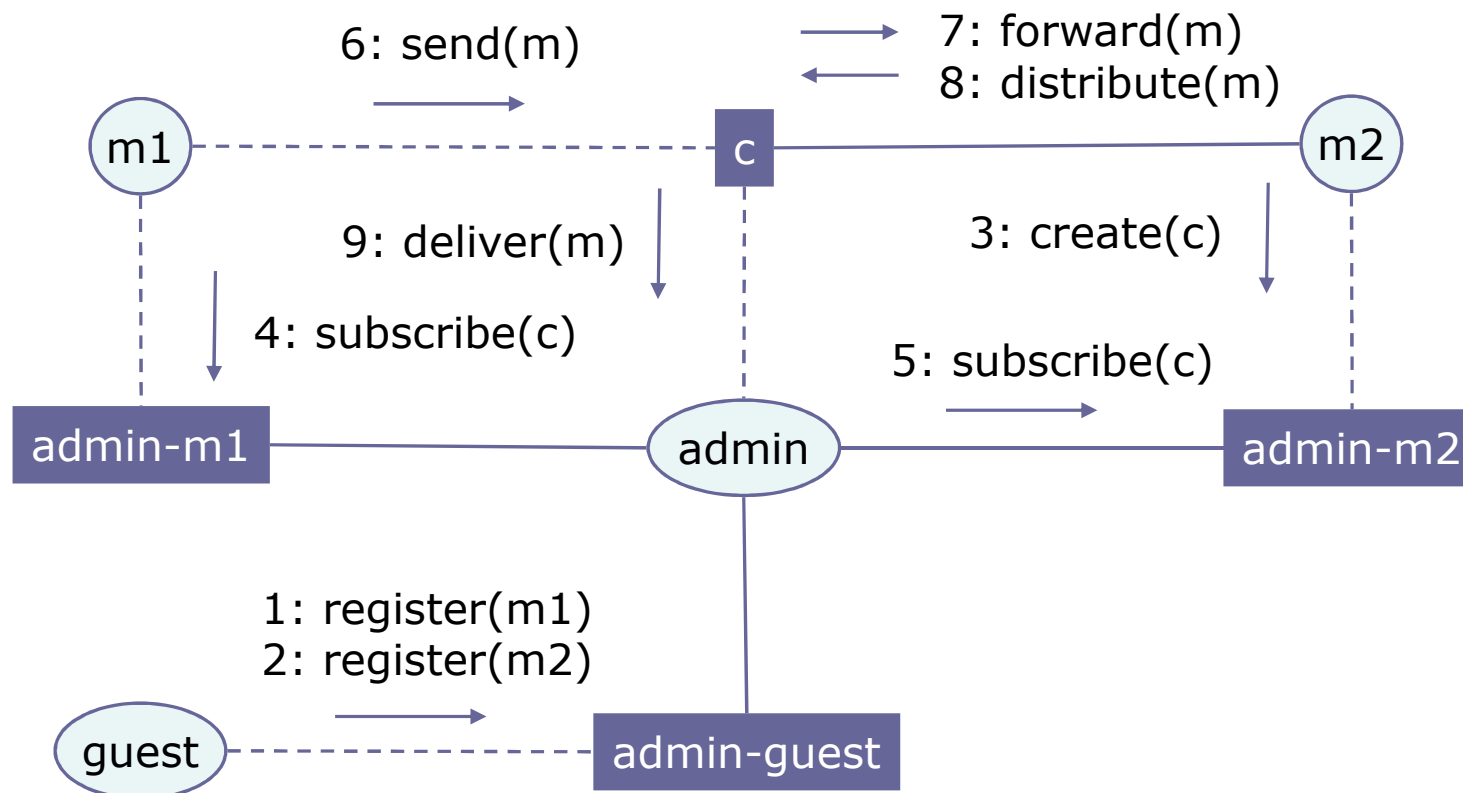
Behavior - Core Messaging 3

Subscribe to a channel:



Behavior - Core Messaging 4

Send and receive a message:

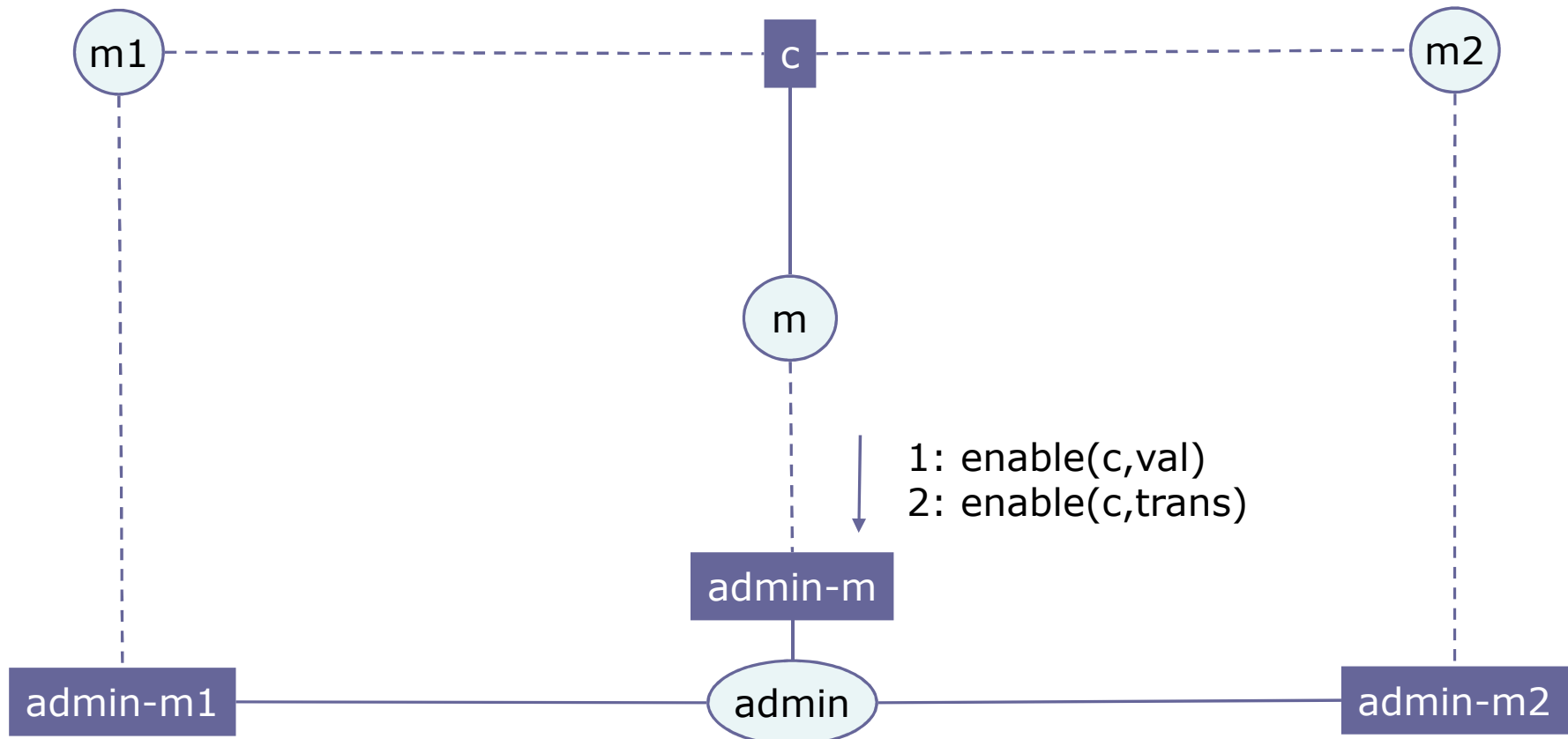


Behavior - Horizontal Extension

Horizontal Extension	1) enable validation extension
	2) enable transformation extension
	3) configure both extensions
	4) use extensions: valid/invalid message

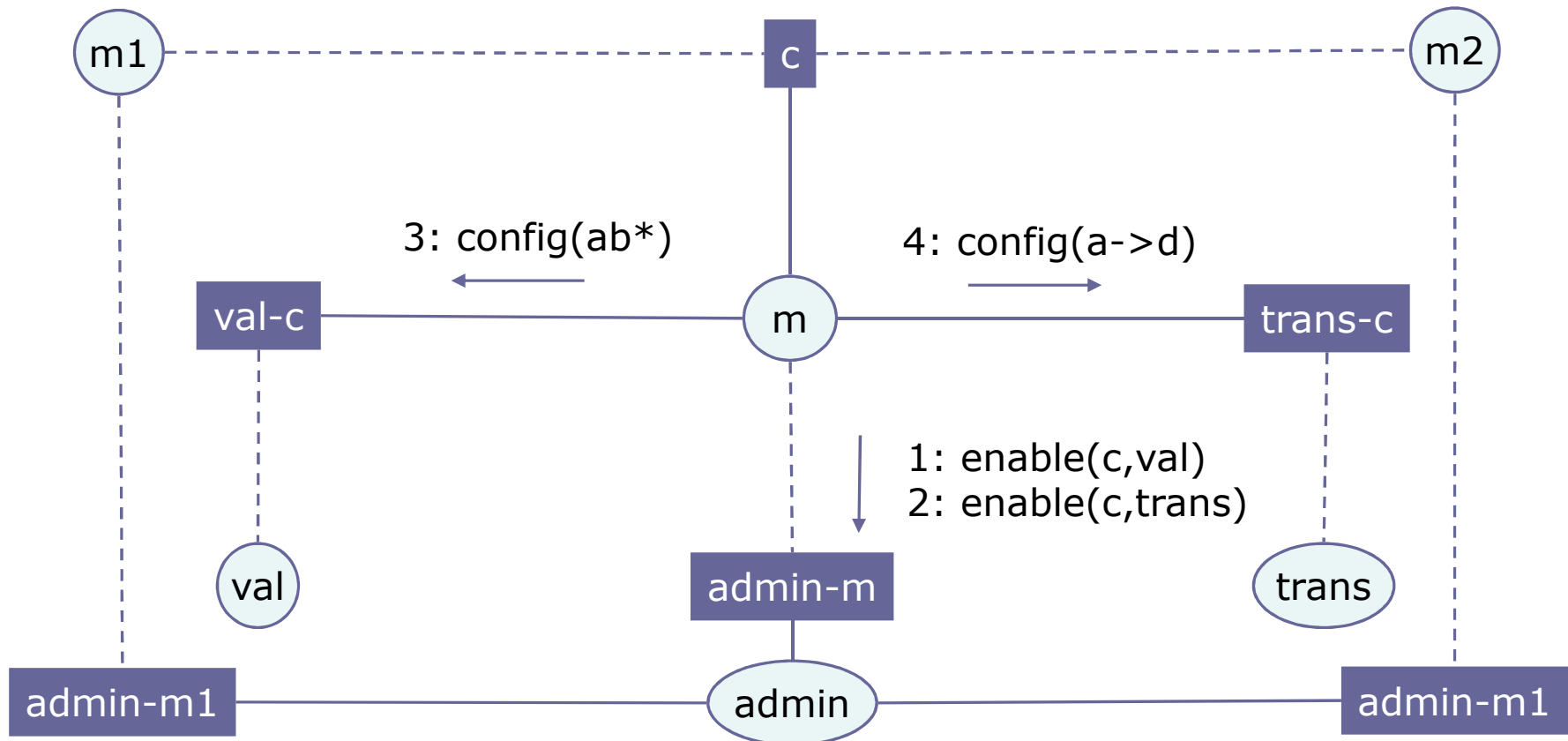
Behavior - Horizontal Extension 1

Enabling validation and transformation extensions:



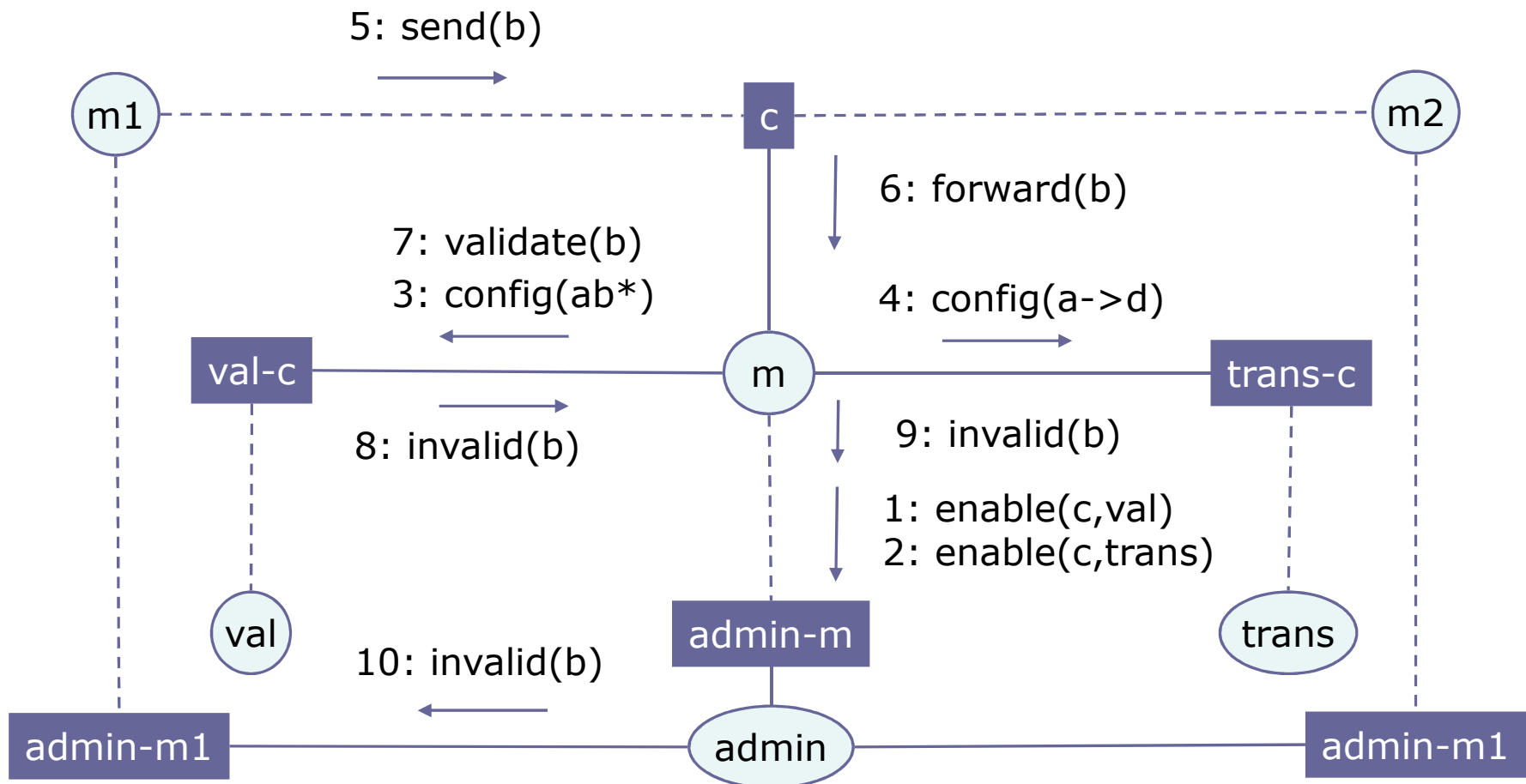
Behavior - Horizontal Extension 2

Configuring validation and transformation extensions:



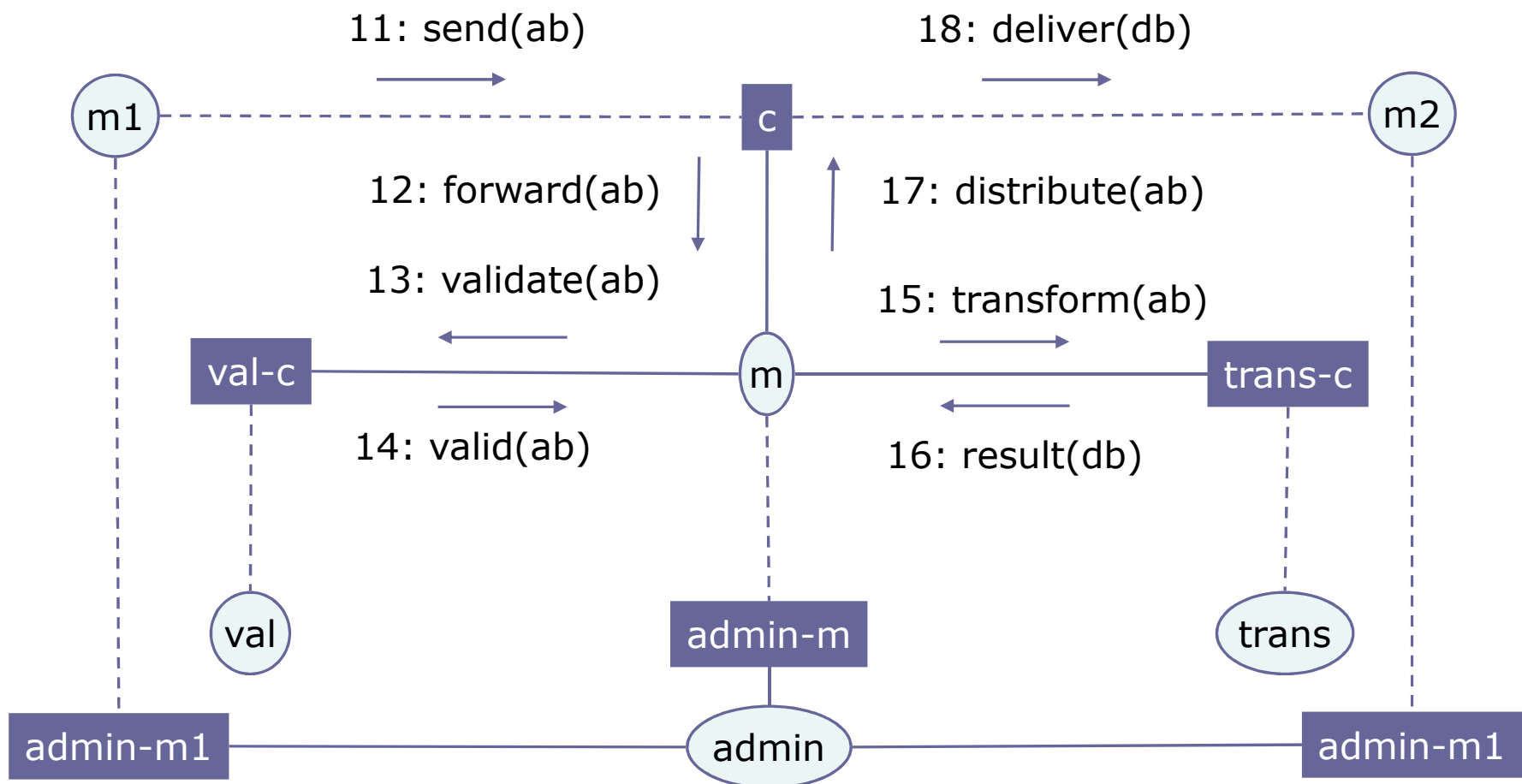
Behavior - Horizontal Extension 3

Using extensions, invalid message:



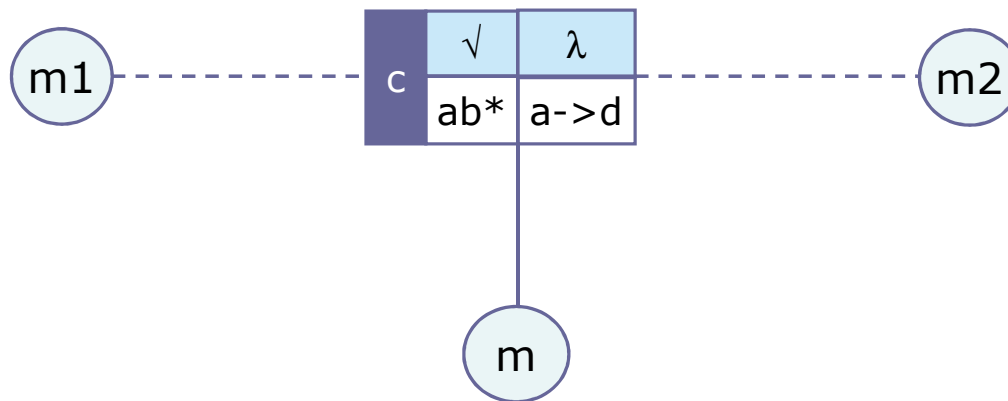
Behavior - Horizontal Extension 4

Using extensions, valid message:



Behavior - Horizontal Abstraction

A channel with validation and transformation extensions:



Extension Repository

Horizontal	Logging	stores messages sent through a channel
	Validation	validates messages with respect to given format
	Transformation	transforms messages from one format to another
	Cryptography	encrypts/decrypts messages in transit
	Authentication	authenticates members to access services
	Directories	enables searching and discovering members
	Member Alliances	enables creating groups of members
Vertical	Process Enforcement	assures correct execution of business processes
	Policy Monitoring	monitors compliance with specified policies
	Channel Composition	enables composition of existing channels
	Tracking	determines the location of a message in transit

Model – State

State	a map of uniquely named variables
Variable	a structure with three mandatory parts (identifier, owner, type)
Structure	a collection of parts with two mandatory parts (identifier and type)
Part	comprises a value and a category
Type	determines the mandatory and optional parts in a structure

type

```

State = Id -m-> Var,
Var = {| s: Struct :- subType(sType(s), vType) |},
Struct = {| s: Category -m-> Part-list :- iswf(s) |},
Part, Category, Type, Value, Id,
NatI == inf | n(Nat)

```

value

```

cat: Part -> Category,
val: Part -> Value,
may, must: Type -> (Category -m-> NatI),
subType: Type >< Type -> Bool,
top: Type :- must(top) = [typeCat +> n(1), idCat +> n(1)],
(all t: Type :- subType(t, top)) ...

```

Model – State Expressions

Capture various ways in which the state can change – syntax and semantics.

type

```
StateX ==
  noChange |
  declare(StructX) | undeclare(VarX) | change(VarX, StructX) |
  seq(StateX, StateX) | con(StateX, StateX) |
  test(BoolX, StateX, StateX), ...
```

```
BoolX ==
  tt | not(BoolX) | and(BoolX, BoolX) |
  hasVar(Id) | hasPart(PartX, StructX) |
  hasType(StructX, Type) | hasValue(PartX, ValueX) | ...
```

value

```
exec: StateX >< State >< State --> Bool
eval: BoolX >< State --> Bool
```

StructX, VarX, BoolX, etc. are expressions that can be evaluated on a state and return the corresponding Struct, Var, Bool, etc. values.

Model – Messaging Structures

Message	a structure with four mandatory parts (id, owner, variable, type)
Box	a variable with any number of message-category parts
Member	an identifier, owns all variables which owner part equals this id the variables owned by a member constitute this member's state every variable is owned by exactly one member every member owns a pair of in-box and out-box variables
Admin	a distinguished member who owns a variable with ids of all members
Channel	a variable with any number of member-id parts (subscribers)

type

```
Mes = { | s: Struct :- subType(sType(s), mType) | },
```

```
Box = { | v: Var :- subType(sType(v), bType) | }
```

value

```
mType : Type :- must(mType) = [ idCat +> n(3), typeCat +> n(1) ],
```

```
bType : Type :- must(bType) = [ idCat +> n(2), mesCat +> inf ... ]
```

Model – Messaging Services

A range of services offered by the gateway.

value

```

register: Id --> (StateX << BoolX),
unregister: Id --> (StateX << BoolX),
create: Id << Id --> (StateX << BoolX),
destroy: Id << Id --> (StateX << BoolX),
subscribe: Id << Id --> (StateX << BoolX),
unsubscribe: Id << Id --> (StateX << BoolX),
send: Id << Mes --> (StateX << BoolX),
receive: Id --> (StateX << BoolX << Mes),
deliver: Id --> (StateX << BoolX)

```

The first expression returned (StateX) determines how the state is modified. The second expression returned (BoolX) returns the precondition for the service.

Model – Semantics of Deliver

How to specify the deliver function abstractly?

Three possible effects:

1. new members are registered
2. existing members are unregistered
3. existing members' states are modified:
 - 3.1. a new variable is created
 - 3.2. an existing variable is destroyed
 - 3.3. an existing variable is modified:
 - 3.3.1. the variable is outbox and a message is added – receive
 - 3.3.2. the variable is inbox and message is added – send
 - 3.3.3. the variable is neither inbox not outbox

Abstraction in G-EEG – High-level specifications are state/variable-based, low-level specifications are action/messaging-based.

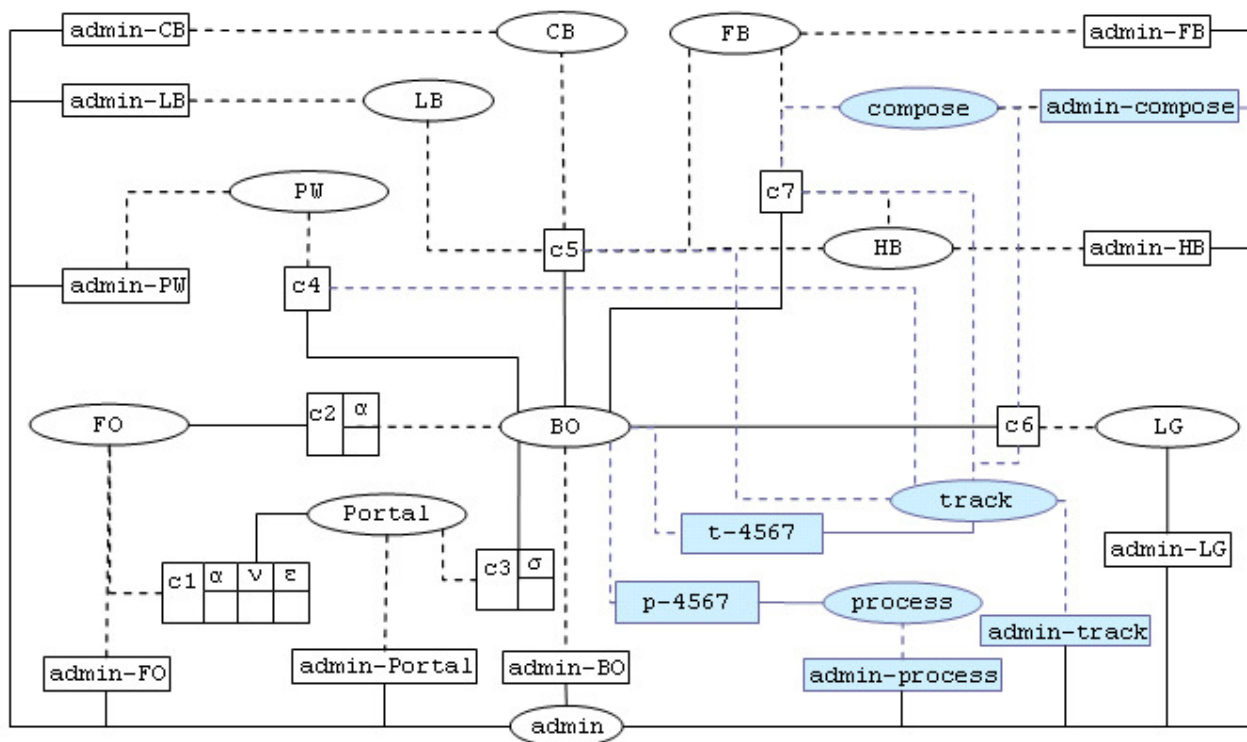
Application – Licensing Process

8 members: Portal, FO, BO, LB, PW, FB, CB, HB

7 channels: C1-C7

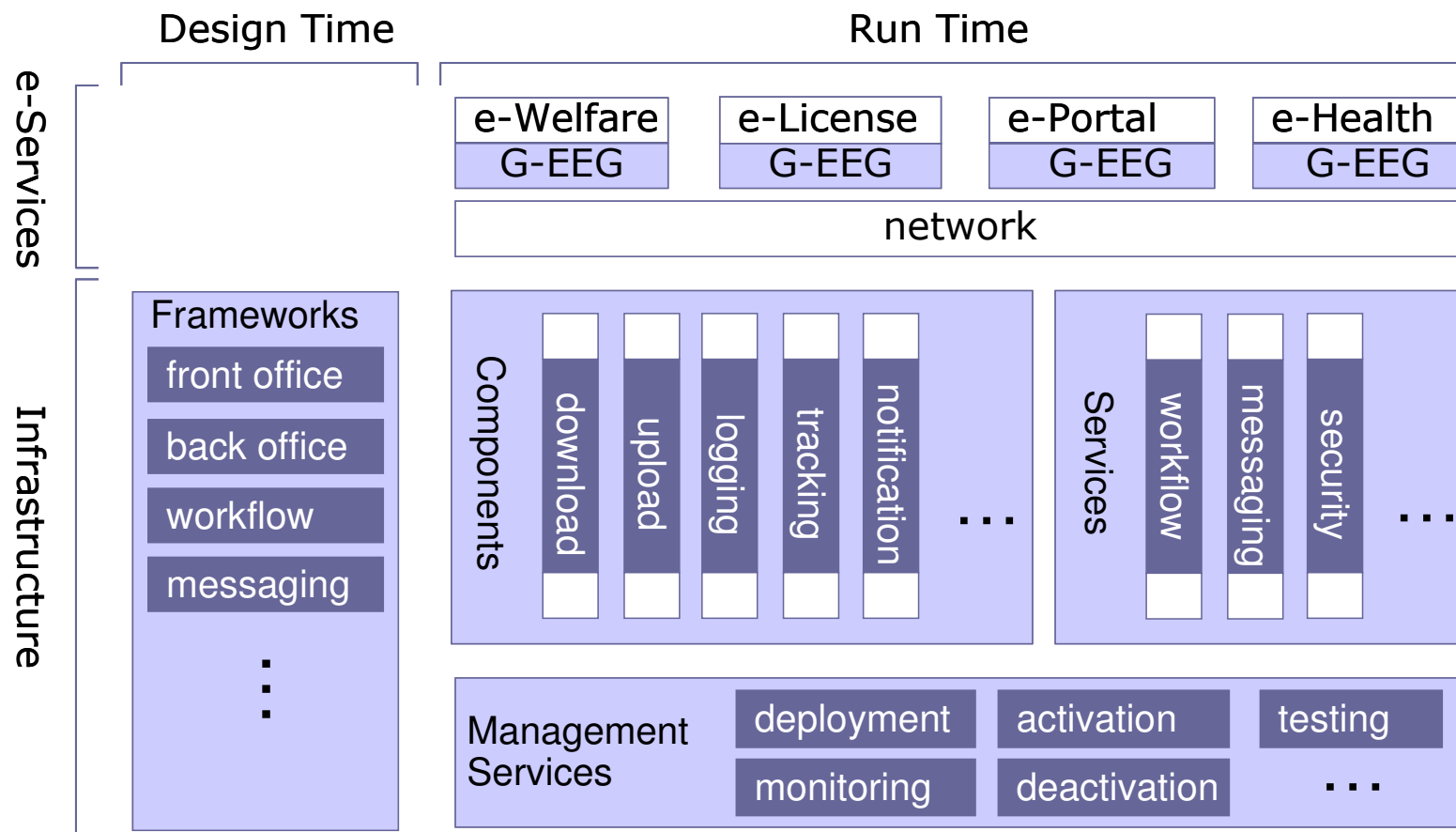
5 horizontal extensions: Logging, Validation, Encryption, Decryption, Auditing

3 vertical extensions: Process-Enforcement, Channel-Composition, Tracking



Implementation - Context

Lightweight software infrastructure for e-Government – enabling rapid development of Electronic Public Services (such as Licensing Service).



Implementation - Technologies

G-EEG-CORE and three horizontal extensions are implemented using:

programming language	Java
object-relational mapping	Hibernate
database	MySQL
messages	eXtensible Markup Language (XML)
XML-Java binding	XMLBeans
validation	XMLSchema
transformation	XSLT

UML was used throughout the development process.

Conclusions

Existing Message-Oriented Middleware does not address many requirements for delivering seamless public services.

G-EEG has been developed to address such requirements.

This work contributes to defining a formal foundations for G-EEG, allowing the definition of formal semantics of messaging services and, in the future, their rigorous development.

Future work: specification, verification and composition of messaging extensions, building a repository of verified extensions, building tools for extension development, product-quality implementation, etc.