Secure-agents

An agent-based simulator for purchases and money withdrawal with an eye on security

Zagnoli Andrea

andrea.zagnoli@studio.unibo.it
II Facoltà di Ingegneria
Cesena

OUTLINE

- Introduction to the secure-agents system
- Why a agent-oriented abstractions and methodologies
- Analysis and design of the system
- Security-related concerns
- Implementation and execution
- Conclusions

INTRODUCTION to the SECURE-AGENTS SYSTEM

Secure-agents

- Simulator which imitates a real-world scenario containing different actors willing to communicate with one another and using sensible resources in a secure manner

- Real-world scenario:
  - online purchases
  - money withdrawal

- Security both for communication while purchasing products and for the process of money withdrawal

Why such a system

- (Maybe)...two birds with a stone
  - SISMA and PROSI

- Employing agent-oriented meta-model and methodology within a coherent, ordered and well defined process of software development (Natali dixit)

- Getting the hands dirty using agent-oriented languages (or frameworks) in order to build such a system

WHY BOTHERING WITH AN AGENT NOTION

Every problem calls for its own level of abstraction

- every problem has its own domain
  - suggests different concepts
- no platform (framework or language) is so general which can directly map those concepts into suitable abstractions of its meta-model
  - there is a conceptual gap between a general problem and the meta-models currently available
- hence, the most suitable software engineering tools for a given problem is the one that employs notions with the minimum semantic gap with respect of the ones proposed by the problem itself [Jennings]

secure-agents domani model (in large)

- resellers and customers as active and interacting entities
- resources with functionalities such as the ATM
- complex communication protocols among actors
- the simulation environment and the simulator environment as interrelated but separated logical spaces

A&A meta-model

- Agents as autonomous computational entities
  - both reactive and pro-active
  - social (i.e. agents speak to one another)
- Weak-notion of agency
  - not proper of the A&A meta-model, but here adopted for simplicity
  - no need for intelligence here (simple and quite static environment)
- Artifacts as purely reactive computational entities designed to support agents' actions towards achieving their goals
  - agents use artifacts and artifacts manifest themselves to the agents
- Workspace as the logical topologic subdivision of the MAS

SODA: AOSE methodology

- Methodology proposes a set of phases, each containing notions at a different level of abstraction, to model the system starting from requirements down to design
- SODA (Societies in Open and Distributed Agent spaces) is an agent- oriented methodology for the analysis and design of agent-based systems, which takes into account both the Agents and Artifacts (A&A) meta-model, and a mechanism to manage the complexity of system description.
  - The methodology is organized in two phases, each structured in two sub-phases: the Analysis phase, which is composed of the Requirements Analysis and the Analysis steps, and the Design phase, which is composed of the Architectural Design and the Detailed Design steps

Requirements

Requirements

- simulation requirements
  - set-up simulation environment and participants
  - launch, pause and stop a simulation-instance
  - inspecting relevant simulation’s data/parameters (public certificates)
  - check minimum requirements for launching the simulation (at least one reseller and one costumer)

- simulation-instance requirements
  - resellers advertise their products
  - costumers wonder around the virtual world, perceiving ads of different products
  - costumers, sometimes, buy products from a particular reseller
  - resellers, when a certain costumer requests it, sell their advertised products
  - costumers, before buying a product, check if they need any money and, if so, they withdraw some from the nearest ATM

Requirements (cont'd)

- **security requirements** (involve and encompass both types of requirements cited before)

- set-up a PK infrastructure in order for agents to communicate in a secure manner
  - *certificate-authority, registration-authority and directory-service*

- communication between a costumer and a reseller has to follow a pre-defined protocol

- the ATM has to employ a mechanism to safely ensure authentication of costumers wanting to withdraw some money

Requirements
(cont'd)

- necessity of adopting measures to prevent unauthorized access to the resources of the system
  - both in terms of access to the virtual resources of the environment and the physical 'secrets' (keys and passwords) used within the system

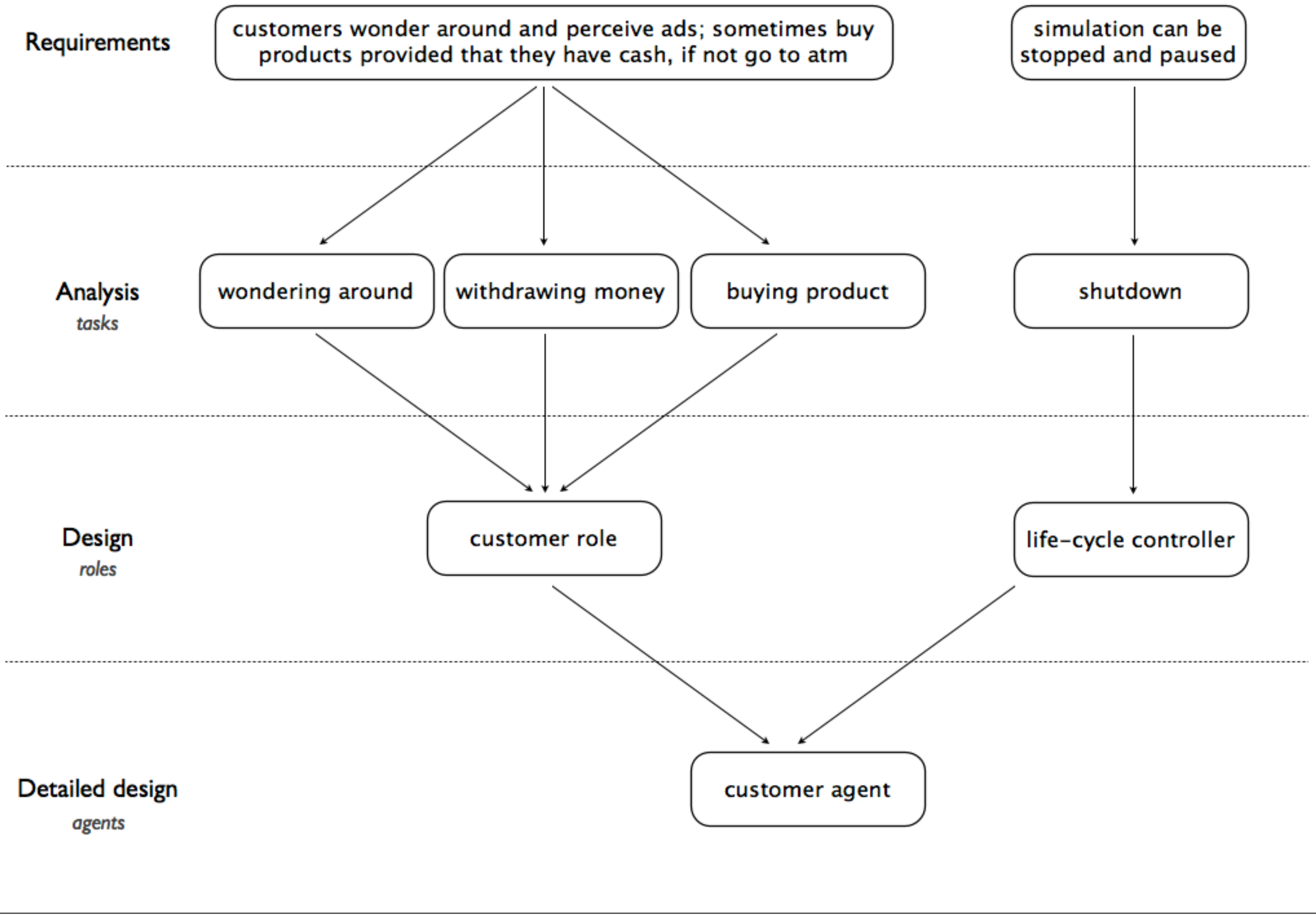
- security requirements both in terms of resources needed and in terms of constraints within the interaction space and in the behavior of the single agents

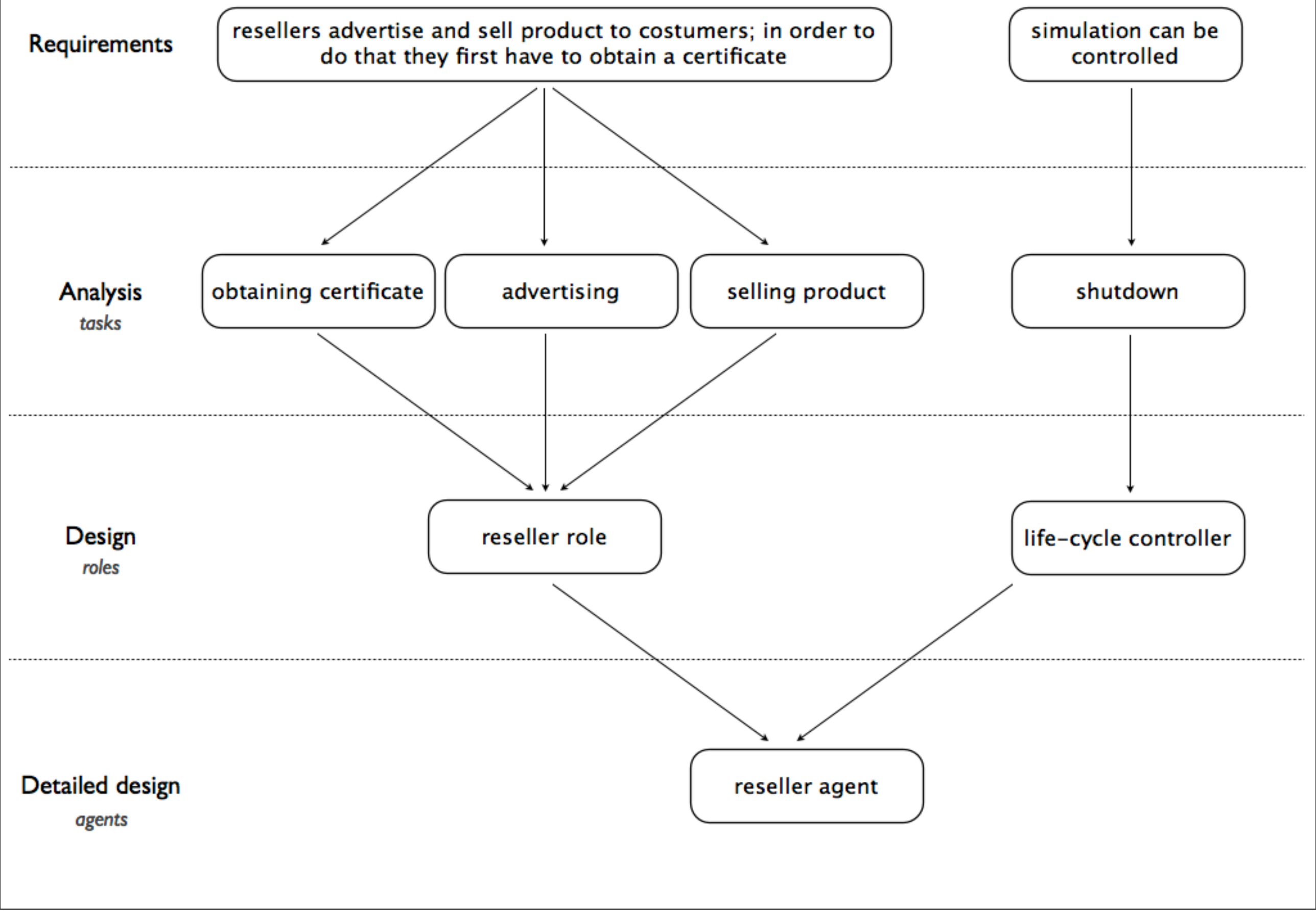
From requirements to detailed design using SODA

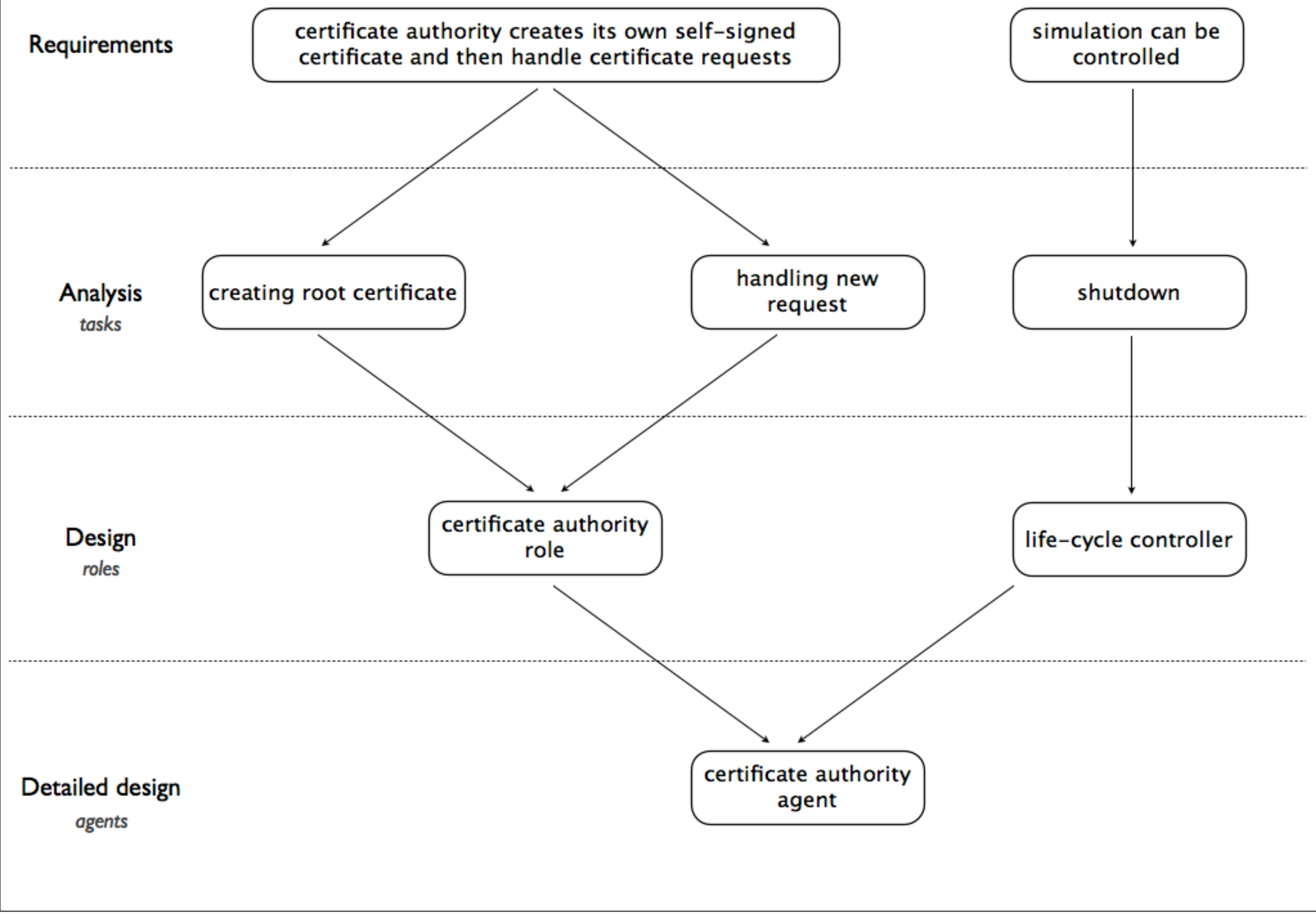
(some examples)

SODA: phases and notions

- Requirements phase
  - actors, requirements and legacy-systems
- Analysis
  - tasks, functions, dependencies and topologies
- Design
  - roles and actions, resources and operations, interactions and rules, spaces
- Detailed design
  - agents, artifacts, workspaces, societies, ...
  - speakTo, manifest, use, ...


Requirements
customers wonder around and perceive ads; sometimes buy products provided that they have cash, if not go to atm
simulation can be stopped and paused
Analysis
tasks
wondering around
withdrawing money
buying product
shutdown
Design
roles
customer role
life-cycle controller
Detailed design
agents
customer agent


Requirements
resellers advertise and sell product to costumers; in order to do that they first have to obtain a certificate
simulation can be controlled
Analysis
tasks
obtaining certificate
advertising
selling product
shutdown
Design
roles
reseller role
life-cycle controller
Detailed design
agents
reseller agent


Requirements
certificate authority creates its own self-signed certificate and then handle certificate requests
simulation can be controlled
Analysis
tasks
creating root certificate
handling new request
shutdown
Design
roles
certificate authority role
life-cycle controller
Detailed design
agents
certificate authority agent

Requirements

an ATM is used by the agents to withdraw money, it has to use the challenge/response mechanism to authenticate clients

Analysis
functionalities and constraints

insert client's badge

insert response

insert badge before insert response

Design
resources and rules

ATM resource

ATM use rule

Detailed design
artifacts and protocols

ATM artifact

ATM artifact's usage protocol

Requirements

each costumer when needs to buy a product from a reseller has to directly contact him using an agreed form of messages

security needed when communicating

Analysis

tasks and constraints

selling products

buying products

s-p and b-p depends on one another

agreed form of messages

Design

roles interactions and rules

reseller role

customer role

reseller role and customer role interact

rule for the form of messages exchanged

Detailed design

agents, speakTo and protocols

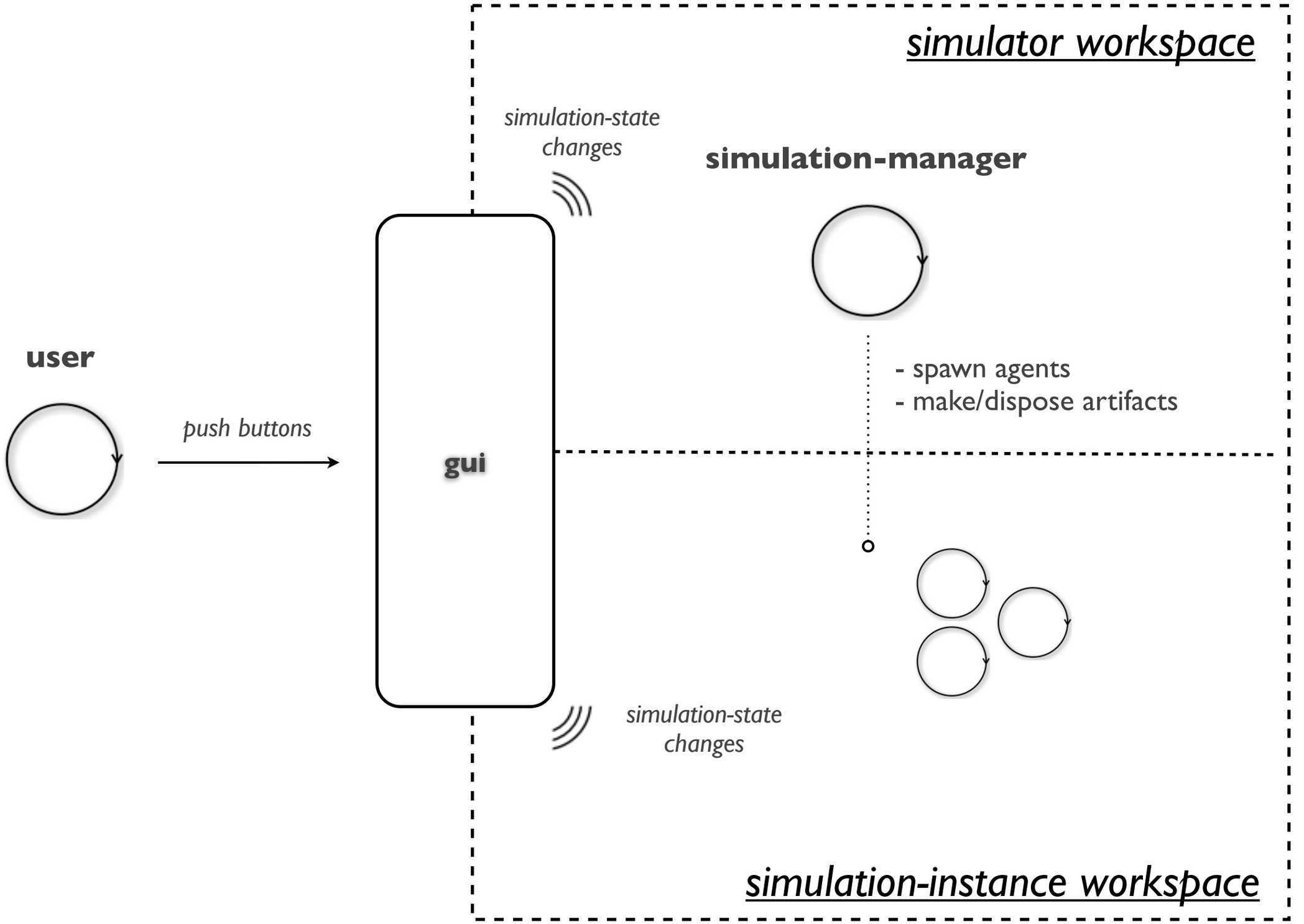
reseller agent

customer agent

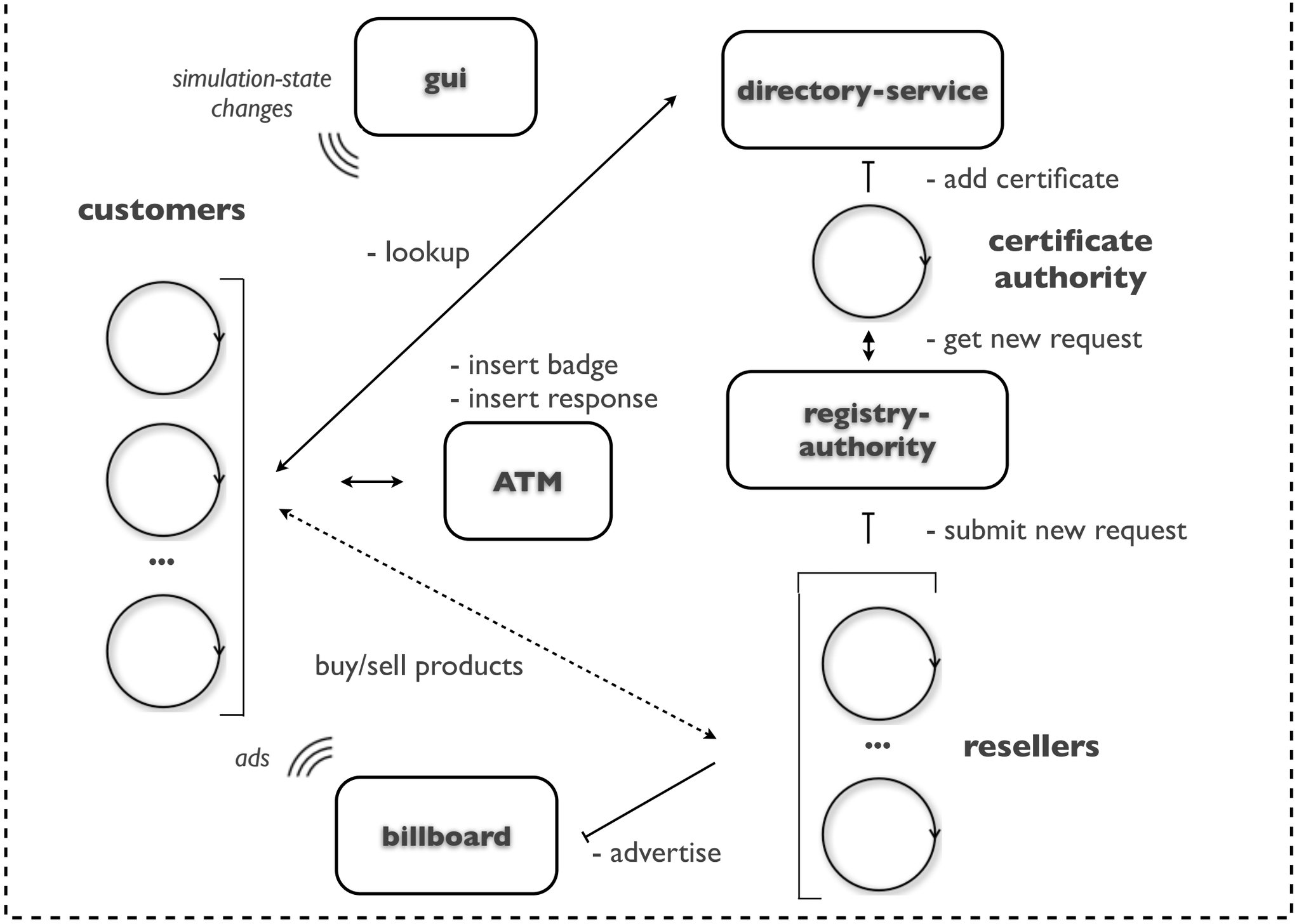
reseller agents and customer agents speak to one another

ATM artifact's usage protocol

secure-agents system



simulation-instance workspace



Security in depth

RBAC

- resources' access control within multi-agent systems
- access decisions depend on the functions the agents are currently performing within the organization, namely their current role within the society
- agents play different roles during their life-cycle
- policy stating what can and cannot be done based on the role
  - permission or denial of a role of performing a certain operation on a resources
  - i.e. entering a workspace or using an artifact (or certain operations)

System's policy

- closed-policy assumption

Role	Artifact [allowed operations]
Customer	ATM [`insert-badge`, `insert-response`] Directory-Service [`look-up`]
Reseller	Registration-Authority [`submit-CSR`] Billboard [`advertise`]
Certificate-authority	Registration-Authority [`get-new-request`] Directory-Service [`addCertificate`, `look-up`] GUI [`submit-new-request`]

Implementation and execution

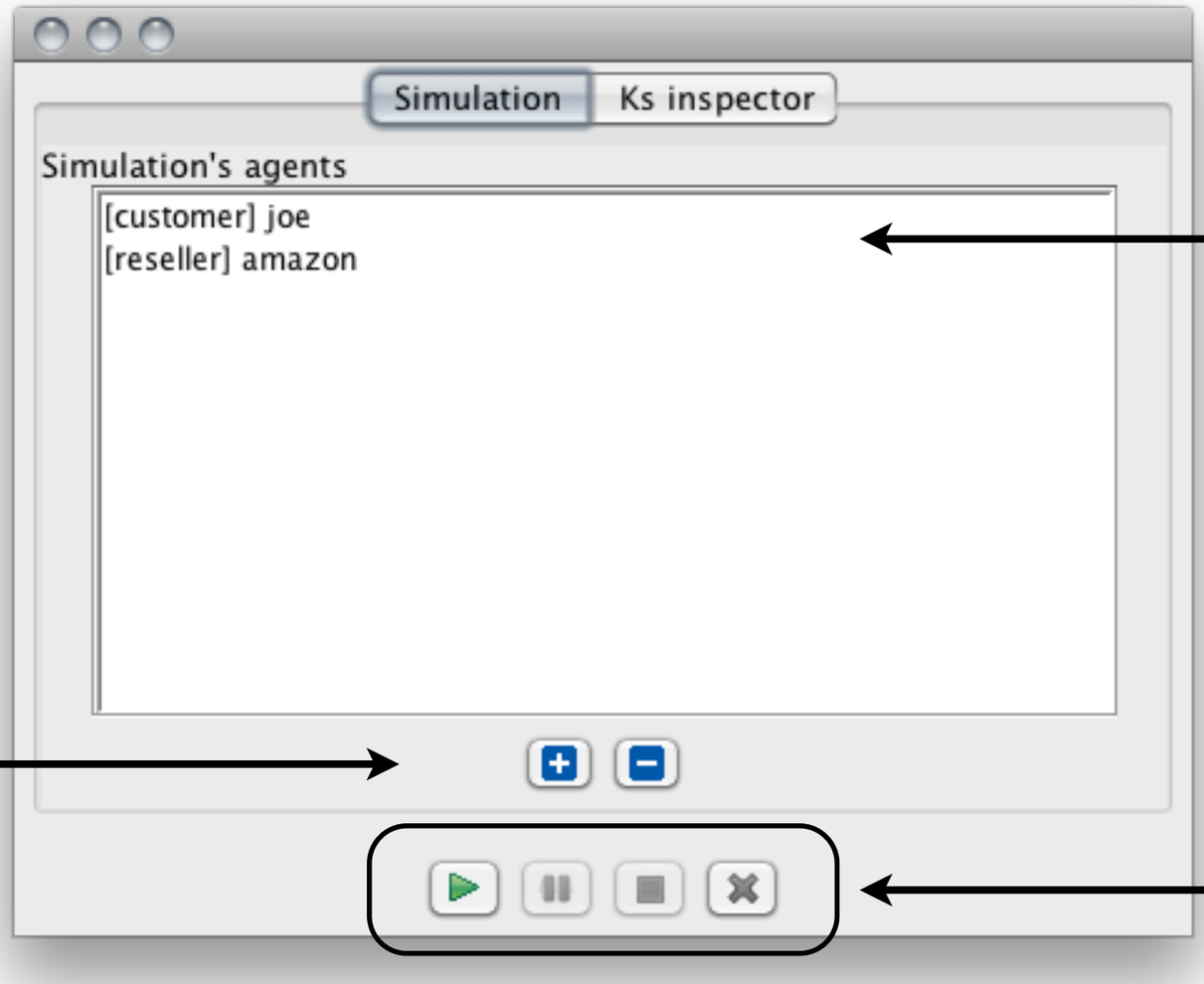
simpA framework

- simpA as platform
  - a annotation-based extension of JAVA
  - written in JAVA
- adopts A&A meta-model
  - active notions of agents, activity, agenda
  - purely reactive notions of artifacts and operations
  - topological notion of workspace

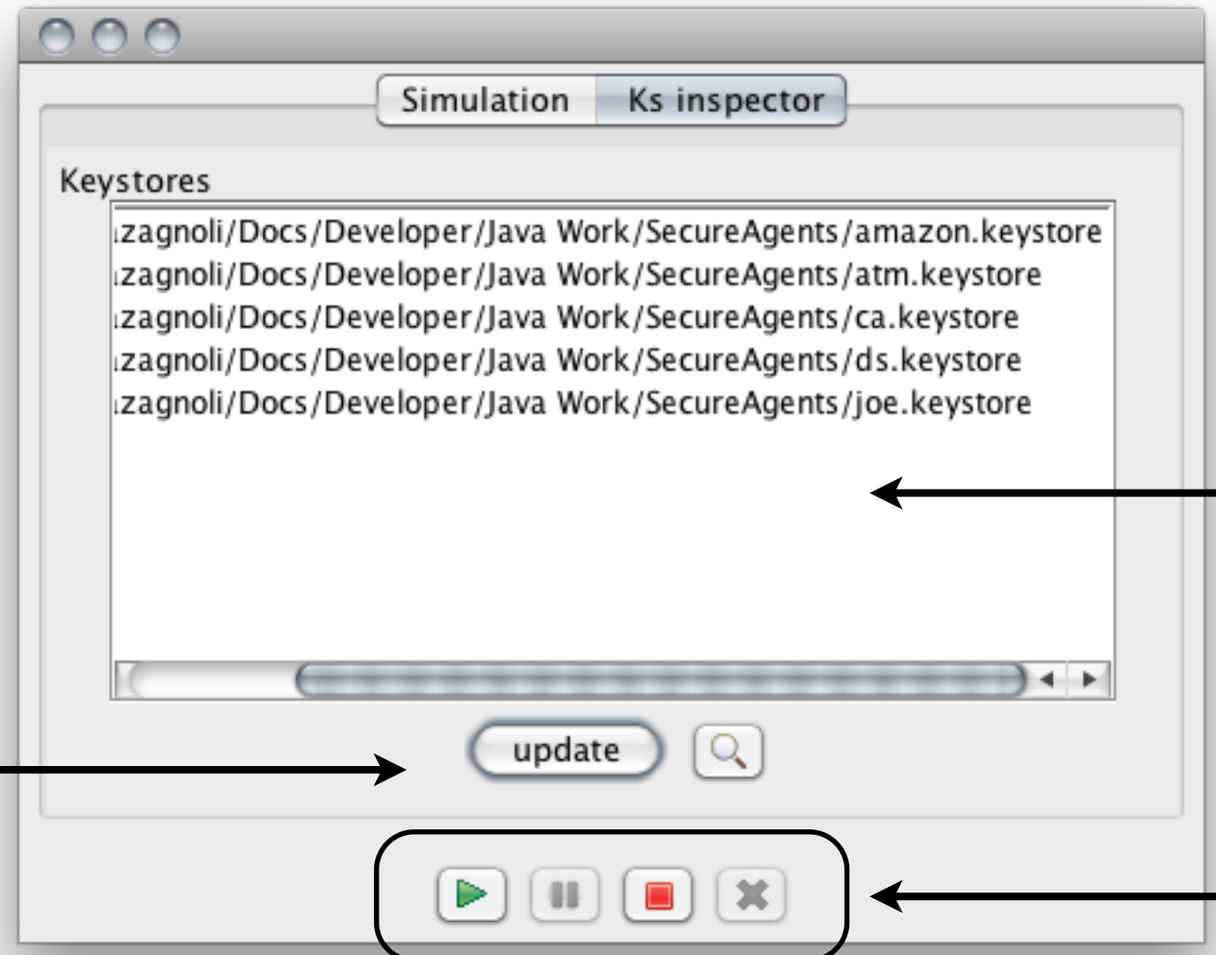
Execution

- executing the secure-agents.jar from the console, within the folder containing it
  - $ java -jar secure-agents.jar
- command-line prints of what is currently happening within the system will appear along with the system main gui

Interface: main window (Simulation environment)



Interface: main window (Key-store inspector)



key-stores

update or inspect a key-store

simulation life-cycle controllers

Conclusions

Notes on the meta-model

- agent abstraction, as the natural extension of modularity and encapsulation, is suited to map system's active parts
  - reducing the semantic gap between the problem and the model of its solution

- artifact are useful to map those parts of the system which are only characterizable by a purely reactive behavior
  - the virtual environment in which agents operate is tailored to their needs through the artifacts

- workspaces as the logical containers and separators of a MAS

Notes on the implementation

- simpA is an agent-oriented framework based on Java adopting the A&A meta-model
  - essentially annotated Java
- pros:
  - pragmatic
  - easy to use (being Java)
- cons:
  - simpA's agent internal architecture completely lacks the balance between proactivity and reactivity, dealing only with the first one and demanding the latter directly to the developer
  - not an ad-hoc language, but an object-oriented extension towards agents

Notes on not properly addressed issues

- some parts of the system have been oversimplified for the sake of simplicity
- communications among agents does not employ any ACL based on some pre-existing and shared ontology
  - the problem of understanding each other not treated
- dynamic constraints in the RBAC
- certificates revocation lists

Secure-agents

An agent-based simulator for purchases and money withdrawal with an eye on security

Zagnoli Andrea

andrea.zagnoli@studio.unibo.it
II Facoltà di Ingegneria
Cesena