# On the formalization of the Airport Domain


BY

IAKOVOS OURANOS

*HCAA, NTUA*


*DECEMBER 2018*

# Outline

- **Background and Motivation**
- The Airport Domain
- Ongoing & Future Work
- Related Work

# Background

**Formal Methods** + **Domain Description** = **Verifiable software**

Domain Engineering

**Need for :**

- Verifiable software
- Bug – free software
- Verifiable designs
- Requirements
- Better understanding of the domains

# Domain Engineering (1)

**Bjorner's triptych dogma**

a) Before software can be developed, the software developers and the clients contracting this software must understand the requirements.

b) Before requirements can be developed the software developers and the clients contracting these requirements must understand the domain.

**Domain Description**

- Document in a natural and formal language
- Describes observable phenomena of the domain (entities, functions, events, behavior)
- If it is written in an executable formal language, validation and verification of the domain description can be performed

# Domain Engineering (2)

## Some Definitions

**Domain**: Universe of discourse, area of human activity, area of science, sufficiently well distinguished from neighboring areas to avoid overlap.

**Example** Air traffic, Health care, Transportation, Financial services activity, etc

**Entities**: Refer to a domain's manifest phenomena. Things that you can point or measure, and concepts derived from these. Can be fixed, immobile or static. There are atomic and composite entities.

**Example** Hospital Domain: Patient, Medical Staff, Admin Staff (Atomic), Operating room, Ward, Pharmacy, Technical Support (Composite)

# Domain Engineering (3)

## Some Definitions (continued)

**Functions**: Are applied to entities. Some of them are input to the domain, while some others are states and/or context values of the domain, and yield entities, changing the state of the domain.

**Example** Hospital Domain: adding a citizen as a patient, create a new medical record, admitting a patient in the hospital, etc.

**Events**: Are the changes of some state or context of the domain under certain conditions. In other words is the occurrence of something that may trigger an action or is triggered by an action, or alter the course of a behaviour or a combination of them.

**Example** Financial Service Industry Domain: The event of going below a credit limit when withdrawing money from an account.

# Domain Engineering (4)

## <u>Some Definitions (continued)</u>

**<u>Behaviours</u>**: Sequences or function actions and events.

**Example** Financial Service Industry Domain: The opening of a demand/deposit account followed by a sequence of zero, one or more deposits and withdrawals and ending with the closing of the accounts.

**<u>Stakeholders</u>**: a person or a group of persons, united somehow in their common interest in, or dependency on the domain; or an institution, enterprise of a group of such, characterised by their common interest in or dependency on the domain

**Example** Government Domain: Politician, Ministry, Parliament, Citizens, Officials, Police are all stakeholders of the Government Domain.

# The OTS/CafeOBJ formal method

- **CafeOBJ** is an algebraic formal specification language.
- **CafeOBJ** is a formal language for writing formal models and reasoning about them with rewritings/reductions.
- **CafeOBJ** is a successor of OBJ and developed by an international team in Japan.
- Related algebraic specification languages:
  1. **Maude** (USA) – Another successor of OBJ
  2. **CASL** (Europe) – Attempt of developing a common algebraic specification language

# The OTS/CafeOBJ formal method

1. **Abstract data types (ADT)** with tight semantics (e.g. integers)
    - Initial algebra semantics
    - Induction based reasoning
2. **Abstract state machines (ASM)** with loose semantics (e.g. objects)
    - Coherent hidden algebra semantics
    - Co – induction based reasoning

# The OTS/CafeOBJ formal method

Two kinds of **sorts**:

- **Visible** sorts representing ADTs.
- **Hidden** sorts representing set of states of an ASM.

Two kinds of **operations** to **hidden sorts**:

- **Actions** that can change a state of an ASM (or object). Takes a state and zero or more data and returns another or the same state.
- **Observations** that are used to observe the value of a data component of an object. Takes a state and zero or more data and returns the value of a data component in the object.

# The OTS/CafeOBJ formal method

## Operator declaration:

- **(action)**

**bop** *action_name* **:** $v\_sort_1$ $v\_sort_2$ … $v\_sort_n$ *h_sort* *h_sort*

- **(observation)**

**op** *observation_name* **:** $v\_sort_1$ … $v\_sort_n$ *h_sort* *v_sort*

## Operator definition with equations:

**eq** $term_1 = term_2$ .

In case of conditional equation:

**ceq** *term1 = term2* **if** $cond_1$ .

# The OTS/CafeOBJ formal method

***OTS*** ***S***: A kind of transition system specified in terms of behavioural specification. It consists of *< O, I, T >* such that:

- $\mathcal{O}$ : A set of observers.

  Each $o \in \mathcal{O}$ is a function $o : \Upsilon \rightarrow D$, where $D$ is a data type.

  $v_1 =_{\mathcal{S}} v_2 \stackrel{\text{def}}{=} \forall o \in \mathcal{O}.o(v_1) = o(v_2)$ .

- $\mathcal{I}$ : A set of initial states.

- $\mathcal{T}$ : A set of conditional transition rules.

  Each $\tau \in \mathcal{T}$ is a function $\tau : \Upsilon/=_{\mathcal{S}} \rightarrow \Upsilon/=_{\mathcal{S}}$ on equivalence classes of $\Upsilon$ wrt $=_{\mathcal{S}}$.

  The condition $c_\tau$ of a transition rule $\tau \in \mathcal{T}$ is called *the effective condition*.

# The OTS/CafeOBJ formal method

An ***execution*** of *S* is an infinite sequence $u_0$, $u_1$,… of states satisfying:

- *Initiation*: $\upsilon_0 \in \mathcal{I}$.
- *Consecution*: For each $i \in \{0, 1, \ldots\}$, $\upsilon_{i+1} =_{\mathcal{S}} \tau(\upsilon_i)$ for some $\tau \in \mathcal{T}$.

A state is called *reachable* wrt $\mathcal{S}$ iff there exists an execution of $\mathcal{S}$ in which the state appears.
Let $\mathcal{R}_{\mathcal{S}}$ be the set of all the reachable states wrt an OTS $\mathcal{S}$.

If predicate $p$ is true in every state of $\mathcal{R}_{\mathcal{S}}$, $p$ is called invariant to $S$, which is defined as follows:

$$\textbf{invariant } p \stackrel{\text{def}}{=} \forall \upsilon \in \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)\,.$$

# The OTS/CafeOBJ formal method

The method includes the following steps:

- A system is modeled as a TOTS.
- The TOTS is written in CafeOBJ.
- Properties to be proved are expressed as CafeOBJ terms.
- Proofs or proof scores showing that the TOTS has properties are written in CafeOBJ.
- The proof scores are verified by executing them with the CafeOBJ system.

# Outline

- Background and Motivation
- **The Airport Domain**
- Future Work
- Related Work

# The Airport Domain (Inf. Descriptions)

- ✓ Entities
- ✓ Functions
- ✓ Events
- ✓ Behaviours

# A. Entities

- ✓Passenger (Atomic)
- ✓Airport Administration (Composite)
- ✓Aeronautical Services (Composite)
- ✓Ground Handling (Composite)
- ✓ Commercial Services (Composite)
- ✓ Catering – F&B Companies
- ✓ Airport/Airline Companies
- ✓ Security  Companies
- ✓ Police – Passport Control Service
- ✓ Fire – fighting service
- ✓ Cleaning service

# B. Entity analysis (Example 1)

**Passenger Entity**

- Atomic Entity
- Interacts with most of the other entities
- Receives information (FIDs – Loudspeakers)
- A combination of Id or Passport – Boarding Pass

# B. Entity analysis (Example 1)

## Passenger Entity Fuctions

- **check in** at a counter desk, or on the web, and **baggage delivery** (interact with a ground handling company and an airline company)
- **preboarding and security/hand baggage check** by the security company
- **passport control** by police officers
- **boarding control and aircraft delivery** by the ground handling and airline company

# B. Entity analysis (Example 1)

## Passenger Entity Fuctions (contd)

- **buy products** from the duty free shops or other commercial companies
- **buy food** from an F&B company (restaurant, bar etc)
- **submit complaints form** to the airport authority
- **declare goods** to the customs
- **receive information**
- **and so on …**

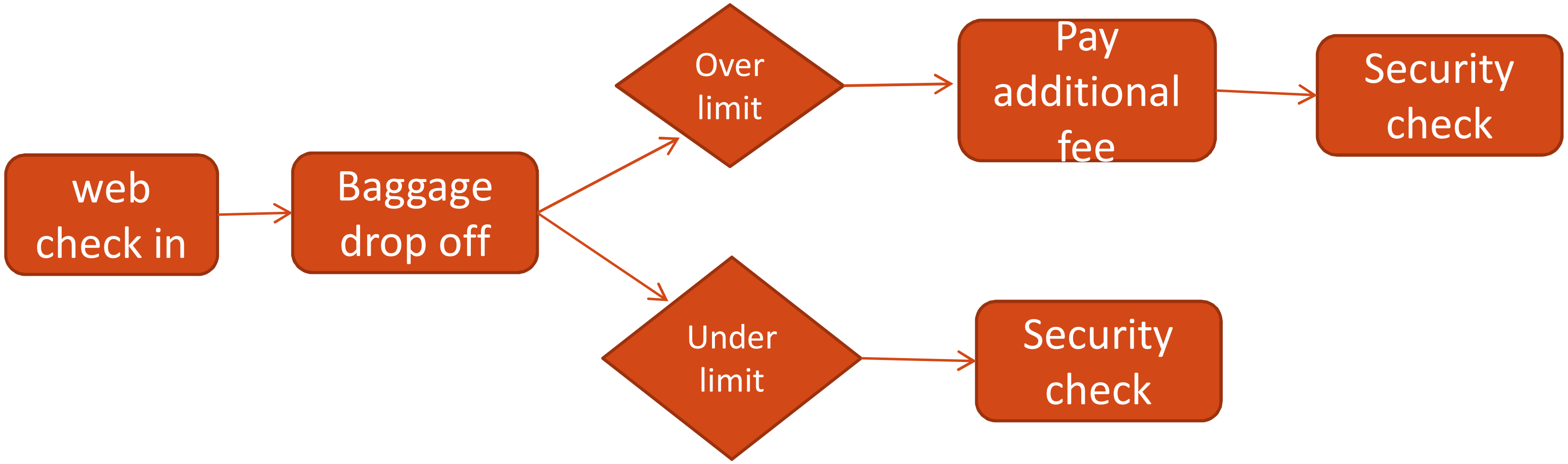
# B. Entity analysis (Example 1)

## Passenger Entity Events

- **Provide or not** valid documents during check in or passport control
- **Deliver restricted or not objects** during security check
- **Provide or not** valid documents during boarding process
- **Having or not** something to declare at the arrival
- **Luggage weights over or under** the allowed limit

# B. Entity analysis (Example 1)

## Passenger Entity Behaviors (intra schengen)



# B. Entity analysis (Example 2)

**Ground Handling Entity**

- Composite Entity
- Interacts with most of the other entities
- Support many critical services of the Airport
- The attribute is the Name of the Company

# B. Entity analysis (Example 2)

## Ground Handling Subentities

- Baggage handling
- Lost and Found
- Check in
- Boarding (Gate support)
- VIP support
- Aircraft support
- Air Cargo

# B. Entity analysis (Example 2)

## Ground Handling Subentities

### Check in entity functions

- check ticket
- check id card or passport
- issue boarding card
- issue baggage tag
- forward baggage on the baggage claim
- reject passenger / reject baggage
- weight baggage
- demand additional fee

# B. Entity analysis (Example 2)

## Ground Handling Subentities

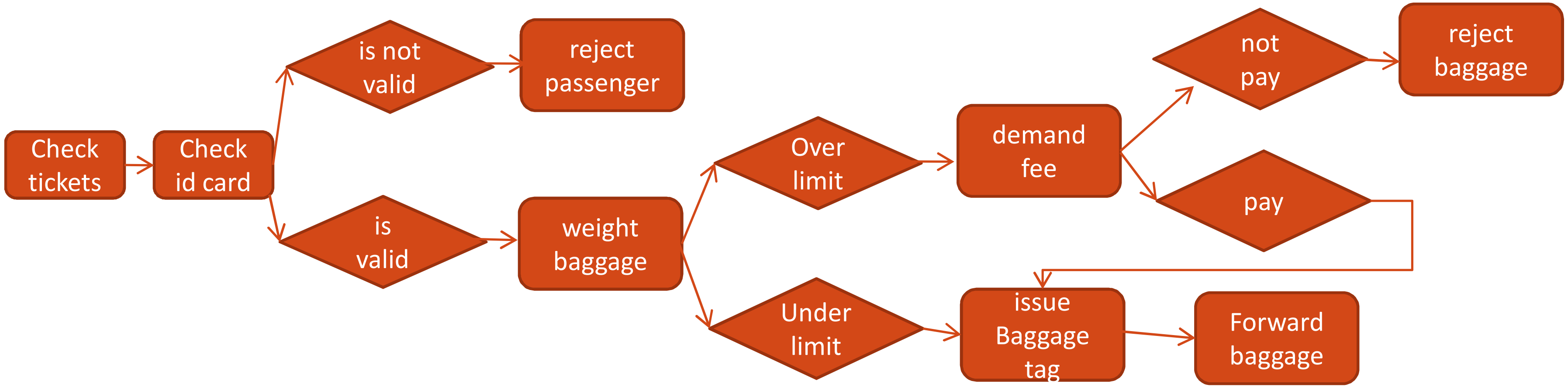
### Check in entity events

- The **provided documents** are / aren’t valid
- The **baggage(s) weight** over/under the limit

# B. Entity analysis (Example 2)

## Ground Handling Subentities

### Check in entity behaviors



# C. Formalize !

- ✓ Our Approach -> The OTS/CafeOBJ method
- ✓ Algebraic Specifications
- ✓A state based method
- ✓ Can be used for specification & validation & verification
- ✓ Suitable method for formalization of DOMAIN DESCRIPTIONS
- ✓ Application to the AIRPORT DOMAIN

# C. Formalize !!

- ✓ Entities can be modeled as initial algebras in CafeOBJ modules (Data Types)
- ✓ A procedure / operation of the Airport can be modeled as an Observational Transition System, a kind of transition system
- ✓ Functions can be modeled as action operators (transitions) of the OTS
- ✓ Events can be modeled as the effective condition of the action operator
- ✓ Behaviors can be modeled as sequences of applications of the action operators

# C. Formalize !!!

## A tiny example

```
-- defining Passenger as a combination
-- of a passport and boarding card.
-- modules defining them have been
-- declared beforehand
mod! PASSENGER {
pr(PASSPORT + BOARDING + EQL)
[Passenger]
op p : Passport Boarding -> Passenger
op ps : Passenger -> Passport
op bd : Passenger -> Boarding
op _=_ : Passenger -> Passenger {comm}
var PS : Passport
var BD : Boarding
vars P P1 P2 : Passport
-- equations
eq ps(p(PS, BD)) = PS . eq bd(p(PS, BD)) = BD .
eq (P = P) = true . eq (P1 = P2) = (ps(P1) = ps(P2) and bd(P1) = bd(P2)) .
}
```

# Outline

- Background and Motivation
- The Airport Domain
- **Ongoing & Future Work**
- Related Work

# Ongoing Work

- Writing complete specifications in CafeOBJ for the Airport Domain for each basic operation
- Prove invariant properties of the written specs

# Future Work

- Model check – Falsify the specs with Maude
- Write rules in RuleML for each operation of the Airport
- Apply the approach to security aspects of Smart Airports

# Outline

- Background and Motivation
- The Airport Domain
- Future Work
- **Related Work**

# Related Work

- Chen, X. (2009) Towards transparent e-government systems - A view from formal methods, PhD Thesis, JAIST, School of Information Science.
- Dinesen A., Alameddine, I. (2000) Towards domain, requirements and software design descriptions for Airport Management Applications

# Thank you!!!!

Iakovos Ouranos

iouranos@gmail.com

QUESTIONS ???