

Title	A Domain Analysis of Artistic Works and A Digital License Language
Author(s)	Xiang, Jianwen; Bjorner, D.; Chen, X.; Arimoto, Y.; Ogata, K.; Vestergaard, R.; Futatsugi, K.
Citation	
Issue Date	2006-11-28
Type	Presentation
Text version	publisher
URL	http://hdl.handle.net/10119/8317
Rights	
Description	3rd VERITE : JAIST/TRUST-AIST/CVS joint workshop on VERification TEchnologyでの発表資料, 開催 : 2006年11月27日 ~ 28日, 開催場所 : JAIST 知識科学研究科講義棟・中講義室

A Domain Analysis of Artistic Works and A Digital License Language

JAIST/TRUST - AIST/CVS
Joint Workshop on Verification Technology 2006

Presenter: Jianwen Xiang
Collaborators: D. Bjørner, X. Chen, Y. Arimoto, K. Ogata,
R. Vestergaard, and K. Futatsugi

Outline

- Why All This ?

- The Domain
 - Entities, Functions,
 - Events, Behaviours
 - Why **Narrate** a Domain
 - Why **Formalise** a Domain

- The License Language
 - The Design
 - Examples
 - Narration and Formalisation

- Closing

Why Should Science & Technology Be Interested in This?

- As **Engineers**
 - Domain engineering **precedes** requirements engineering and software design.
- As **Scientists**
 - So we need to research and develop possible **principles**, **techniques** and **tools** for domain engineering-based software development.
- As **Technologists**
 - To further strengthen **software house capabilities** (professionalism) and to prepare for **product line management**.

The Artistic Works Domain: Entities, Functions, Events, Behaviours

Analysis: Basic Entities

- The intrinsic entities of the performing arts are the **artistic works** themselves:
 - drama or opera performances,
 - music performances,
 - readings of poems,
 - mpvies, etc.
- We shall limit our span to the scope of electronic renditions of these artistic works:
 - videos,
 - CDs or other.



Analysis: Work Kinds

- Works are either created, rendered, edited, copied or redistributed.
- One can claim that a work can
 - either only (say: “**most recently**”) be an **original**,
 - or only be **rendered**,
 - or only be an edited work (i.e., a **version**),
 - or only be a **master** work which has been copied,
 - or only be a **copy** of a master work,
 - or only be **paid**,
 - or only be **redistributed** (moved to different owners).
- From a work one can observe its **most recent information**, likewise how it has been delivered and consumed.

Narrative: Work Kinds

Thus works are “most recently” either

- created, i.e., they are [blank] “originals”,
- rendered,
- edited,
- the basis for copying, i.e., they are “masters”, or
- the result of copying, i.e., they are copies,
- paid,
- redistributed (further licensed),
- etcetera

Basic Functions

Narrative: Basic Functions

- Works can be
 1. created,
 2. rendered,
 3. copied,
 4. edited,
 5. paid,
 6. licensed,
 7. and a few other things (...).

- Copying a work w results in a pair, (w', w'') :
 1. w' is the basis for copying, i.e., it is a “master”;
 2. w'' is the result of copying, i.e., it is a copy.

Basic Behaviours: Work Histories

Analysis:

- the **pragmatics** behind is so that we can **reason about works**:
 - “That work has been read, by such and such, several times.”
 - “That work is an edited version of a work which we was read from a work which was copied from ..., etc.”
- We wish to also be able to reason the **most recent operations** performed on the work, and about the work as it was before such an operation: “the work from which it resulted”.

Narrative: Work History

1. We can think of a **work history** to be a **list of pairs**: the operation performed and the work.
2. An **original** work has **no pre-history**. It has the history that it was created.
3. A work which has “just” been **rendered** has the pre-history of the work before it was “just” rendered.
4. The two works which have been partaken in a **copying** operation: the master and the copy works have **different “most recent” histories**, but the **same pre-history**:
 - a) The **master copy** has, as “most recent” history that it served as a **master for a copying action**.
 - b) The **copy** has, as “most recent” history that it was **the result of a copying action**.
5.



Why Narrate and Why Formalise a Domain ?

- Why **Narrate** a Domain ?
 - **Stakeholders** must “**sign-off**”!
 - Hence must **understand the description**.
 - Hence the description must also be **informal**.
- Beneficiary **Side-effect**:
 - Domain description can be used for artistic works domain **staff and customer training**.
 - Perhaps even as **school/college training/education material**.
- So we have **informally** described “the” artistic works domain.

Why Narrate and Why Formalise a Domain ? (cont.)

- Why **Formalise** that Domain ?
 - To **understand**.
 - To **analyse**:
 - **Properties**.
 - Contribute to an artistic works **domain theory**.
 - As a **base** for **other theories** related to the domain:
 - **Game theory & practice**: what is possible, transaction costs, etc.
 - **Operations research theory & practice**: scheduling and allocation.

Formalisation: Work Kinds

– An Excerpt of CafeOBJ Spec.

```
[ Work ]
```

```
[ OWrk RWrk EWrk MWrk CWrk PWrk LWrk < Work ]
```

```
op obsWrkOKind : Work -> OKind
```

```
ops og rd ed mw cw pd lc : -> OKind
```

```
var W : Work var O : Work var R : RWrk var E : EWrk
```

```
var M : MWrk var C : CWrk var P : PWrk var L : LWrk
```

```
eq obsWrkOKind(O) = og .
```

```
eq obsWrkOKind(R) = rd .
```

```
eq obsWrkOKind(E) = ed .
```

```
eq obsWrkOKind(M) = mw .
```

```
eq obsWrkOKind(C) = cw .
```

```
eq obsWrkOKind(P) = pd .
```

```
eq obsWrkOKind(L) = lc .
```

Formalisation: Basic Functions

– An Excerpt of CafeOBJ Spec.

[MCWork] -- a pair of MWrk and CWrk

op mw : MCWork -> MWrk

op cw : MCWork -> CWrk

op create : -> OWrk

op render : Work -> RWrk

op copy : Work -> MCWork

op edit : Work -> EWrk

op pay : Work -> PWrk

op license : Work -> LWrk

Formalisation: Work History

– An Excerpt of CafeOBJ Spec.

```
[ History ] : Work-list
op _,_ : Work Work-list -> Work-list
[ Work < Chaos ]
op chaos : -> Chaos
op obsH : Work -> History      op obsPre : Work -> Work

vars W : Work
eq obsPre(create) = chaos .
eq obsPre(render(W)) = W .
eq obsPre(edit(W)) = W .
eq obsPre(mw(copy(W))) = W .
eq obsPre(cw(copy(W))) = W .
...
ceq obsH(W) = W , nil if obsWrkOKind(W) = og .
ceq obsH(W) = W , obsH(obsPre(W)) if not obsWrkOKind(W) = og .
```

So we have **formally** described “the” artistic works domain in CafeOBJ.

Analysis of Licenses

Basic Concept of Licenses

- By a license we shall understand
 - “a **permission** granted by **competent authority** to engage in a business or occupation or in an **activity** otherwise **unlawful**”, as well as
 - “a document, plate, or tag evidencing a license granted”
 - – ‘**Merriam–Webster OnLine**’ .

■ An Example

license l : licensor grants licensee work w
with permitted actions $\{a_1, a_2, \dots, a_n\}$
and obligated actions $\{a_1, a_2, \dots, a_m\}$

License Languages

■ What is a License Language ?

- A **license** is a, possibly electronic, document
 - which is issued by a **licensor** to a **licensee** and
 - which **permits** or **obligates** the licensee to perform certain **actions**
 - on **licensed products** (here, digital artistic works)
- A **license language** therefore defines
 - the **syntax**,
 - the **semantics**,
 - and the **pragmatics**of licenses.
- Especially it defines **which actions are possible**.
- We will now detail a concept of an Digital Artistic Works License Language (DigitALLL).

Analysis of Permissions and Obligations

- Conventional Understanding (XrML, ODRL etc.)
 - **Permissions:** render, copy, edit, license, etc.
 - **Obligations:** pay etc.

- Three Simple Examples
 - **Freeware donation**
 - Donation (paying) for the freeware is permitted rather than obligated
 - **Optional permissions by paying**
 - Basic permission: render
 - Optional permissions: copy and edit by paying (also permitted rather than obligated) an amount of money
 - **Obligated editing and sub licensing**
 - Subtitle language localization by branches (e.g., movies)
 - Official document reviewing and patients treatment (in the Public Administration and Hospital Domains)

Formalisations:

Permissions and Obligations

```
op _/_ : Action Time -> Event
[ Permissions Obligations ] : Event-set
```

```
[ License ]
op permitted : Event License -> Bool
op obligated : Event License -> Bool
op obsPs : License -> Permissions
op obsOs : License -> Obligations
```

```
var E : Event          var L : License
eq permitted(E, L) = E in obsPs(L) or E in obsOs(L) .
eq obligated(E, L) = E in obsOs(L) .
ceq permitted(E) = true if obligated(E, L) .
```



Analysis of Properties of Licenses

- Definition 3.2.1 (Behaviour) A behaviour is a finite sequence of events which occur at distinct times and are ordered by times.
- Definition 3.2.2 (Fulfilled) A license is fulfilled by a behaviour if and only if all the **obligated events** of the license exist in the behaviour.
- Definition 3.2.3 (Valid) A license is valid at a specific observing time with respect to a behaviour, if and only if for all the obligated events of the license whose times are **less than or equal to** the observing time are fulfilled by the behaviour.
- Definition 3.2.4 (Breached) A license is breached by a behaviour at a specific observing time, if and only if the license is **not valid** at the observing time.

Formalisations: Properties of Licenses

```
[ Behaviour ] : Event-list
op fulfilled : License Behaviour -> Bool
op breached  : License Behaviour Time -> Bool
op valid     : License Behaviour Time -> Bool

op _<=_ : Set Set -> Bool
op list2set : List -> Set
op subBefore : Event-set Time -> Event-set

var L : License      var B : Behaviour      var T : Time

eq fulfilled(L, B) = obsOs(L) <= list2set(B) .
eq valid(L, B, T) = subBefore(obsOs(L), T) <= list2set(B) .
eq breached(L, B, T) = not valid(L, B, T) .
```

Other Issues

- Distinct Actions vs. Implied Permissions
 - Exercising the **copy** right **need not to render** the work;
 - `Permitted(edit) implies permitted(copy) = false`
 - Cf. “**edit and save as**” and “**edit and save on**”
 - `permitted(edit) implies permitted(render) = true`
- **Work Re-distribution** vs. **First-Sale Doctrine** (see report)
- **Fair Use Concept** in License and License Languages (see report)
- **Streaming** and **Non-Streaming Media** (see report)

A General Artistic License Language

The Syntax (semi-formal)

■ Licenses

0. [L Ln Lic]
1. [Nm]
2. [Action]
3. [Permissions Obligations] : Action-set
4. op `_:_` : Ln Lic -> L
5. op `mkLic` : Nm Nm Work Permissions Obligations -> Lic
6. op `render` : Permissions Obligations -> Action
7. op `edit` : Permissions Obligations -> Action
8. op `copy` : Permissions{mw} Obligations(mw)
Permissions{cw} Obligations{cw} -> Action
9. op `license` : Nm{lee} Permissions Obligations -> Action
10. op `pay` : Permissions Obligations -> Action

Operations of Licenses

- The operations that the licensees may want to perform on the works.

11. op rnd : Work Ln Permissions Obligations -> RWrk&L

12. op cpy : Work Ln Permissions Obligations
Permissions Obligations -> MWrk&CWrk&L&L

13. op edt : Work Ln Permissions Obligations -> EWrk&L

14. op public : Work Nm{lee} Permissions
Obligations Ln -> LWrk&L

15. op pay : Work Ln Permissions Obligations -> PWrk&L

Example 1:

Licenses for streaming and Non-Streaming Works

- Suppose that a content provider, **p**, produces a new movie, **m**.
- The movie **m** has two forms:
 - an on-demand streaming media with a (net) reference, say **ms**,
 - and a non-streaming media with a file name, say **mf**.
- Suppose that consumers **c1** and **c2** get two different licenses, **l1** and **l2**, for **ms** and **mf**, respectively.

```
l1 : mkLic(p, c1, ms,  
          {render, copy({render}, os1')}, os1)
```

```
l2 : mkLic(p, c2, mf, {render}, os2)
```

```
copy(ms, l1) → l1' : mkLic(p, c1, mf', {render}, os1')
```

Example 2: License for Work Review

Continue Example 1, we suppose that

- the movie producer wants to distribute the movie (file) **mf** to a reviewer, say **r**, for review.
- **r** is allowed to **render** and **edit** (write comments to) the work, and then send the edited work back (**re-distribute**) to **p**.

```
l3 : mkLic(p, r, mf, {render, edit,  
    license(p, {render, edit}, os3')}, os3)
```

```
1) edt(mf, l3)
```

```
2) subLic(mf, p, {render, edit}, os3', l3)
```

```
l3 : mkLic(r, p, mf, {render, edit}, os3')
```

Example 3: License for Work Resell

- Suppose that the movie producer (**p**) delivers the movie (**mf**) to a retailer (**rt**),
- the license (**ln4**): the retailer can **render** (optional), **copy**, and then **sublicense** (resell) the copies to consumers,
- the **copies** sold to the consumers can **only be rendered** and **cannot be further copied**.

```
(a) l4 : mkLic(p, rt, mf, {  
(b) render,  
(c) copy({render, license(anyone, {render}, os4'')},  
         os4')}, os4)
```

```
(a') l4s : mkLic(p, rt, mf,  
(c') {copy({render, license(anyone, {render}, os4'')},  
         os4')}, os4)
```

License Transition Systems

```
mod* LICSYS {
  *[ Sys ]* -- system state
  op init : -> Sys -- initial system state

  -- behaviour observations
  bop oLicensor : Sys Ln -> Nm
  bop oLicensee : Sys Ln -> Nm
  bop oWork : Sys Ln -> Work
  bop oPermissions : Sys Ln -> Permissions
  bop oObligations : Sys Ln -> Obligations
  -- transition rules
  bop tRun : Action Work Ln Sys -> Sys

  -- effective condition for tRuns (transition rules)
  op con-tRun : Action Work Ln Sys -> Bool
  ... }
```

Licenses Transition Systems (cont.)

```
eq con-tRun(A, W, L, S) =
  A in oPermissions(S, L) or A in oObligations(S, L) .

ceq oLicensor(tRun(render, W, L, S), L')
  = oLicensor(S, L') if con-tRun(render, W, L, S) .
ceq oWork(tRun(render, W, L, S), L')
  = (if L = L' then rnd(W) else oWork(S, L') fi)
    if con-tRun(render, W, L, S) .
ceq oLicensor(tRun(license(N), W, L, S), L')
  = (if L = L' then oLicensee(S, L) else oLicensor(L') fi)
    if con-tRun(license(N), W, L, S) .

...
-- axiom for that condition does not hold, works for
  actions
ceq tRun(A, W, L, S) = S if not con-tRun(A, W, L, S) .
```

Remarks of Formal DigitALLL

- Compared with XML-based RELs
 - Formal Semantics
 - Formal Reasoning
 - Executable Formal Specifications (CafeOBJ)
 - Observational Transition Systems (OTS/CafeOBJ)
 - Existence of desirable behaviours
 - Non-Existence of unwanted behaviours
- A Software Engineering Triptych
 - Domain Engineering: domain descriptions
 - Requirements Engineering: requirements prescriptions
 - Software Design: code and documents

What Has Not Been Covered

- Licenses/Works Composition and Decomposition
- More Serious Temporal Analysis
 - Temporal Logic (TL)
 - Formal Reasoning based on that TL
- Constrained Actions
 - Restrictions of actions, e.g., time, device, and so on
 - Could be encoded as parameters of actions
- Etcetera

Questions?

- Please access our homepage
 - Language Design Lab
 - Digital Rights: Consumers and Producers in A Digital World
 - <http://www.idl.jaist.ac.jp/drcp/>

- Any comments are appreciated!