Pattern Based Software Development

Rui Couto
António Nestor Ribeiro
José Creissac Campos

Outline

- Introduction
- Thesis
- The SCARP Approach
- The uCat Tool
- Validation
- Conclusions and future work
- Contributions
- Publications

Introduction

- Several methodologies support the software development process.
- Model Driven Architecture (MDA) proposes a model driven development methodology.
- Process is based on the definition and transformation of architectural models.
- Systematic transformations support the software development.
- Several works support the automation of the transformation processes.

Introduction

- MDA process starts with architectural models.
- Transformation of requirement into architectural models requires a manual transformation process.
- Manual processes are acknowledged for introducing errors and subjectivity.
- A framework is proposed in order to mitigate errors resulting from the manual transformation process.
- As a result, requirement models are included as part of the MDA standard process.

Thesis

- *It is possible to create a framework, based on the MDA and support its extension, by including requirement models transformation as part of the process. Functional requirements, specifically use cases, have the required properties in order to be formalized and support automatic transformation techniques, which support their integration in MDA.*

Proposal

- The proposal is supported by a framework, sustained by a controlled natural language.

- The specifications are converted into an intermediary representation in order to identify requirement patterns.

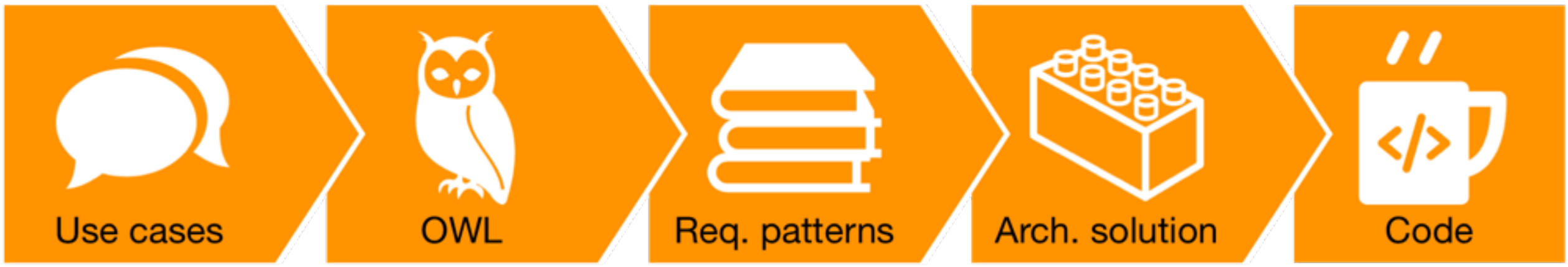
- From requirement patterns, software patterns are selected.

- Software patterns are combined to produce architectural models.

- Architectural models are integrated in the MDA process, producing source code, user interface prototypes, among other analysis artifacts.

The SCARP Approach

- Scenario Based Rapid Software Prototyping (SCARP) is proposed in order to automate the transformation of requirement models into architectural models.



- A tool (uCat) was developed in order to support SCARP.

The SCARP Approach - use cases

- Use cases are formalized through a Controlled Natural Language (CNL), the Restricted Use Cases Statements (RUS)

- Based on a triple format, supports automatic transformation into an OWL representation

User Input	System Response
user selects the login	
	system requests the username and password
user provides the fields	
	system verifies the fields
	system creates a session

The SCARP Approach - OWL

- RUS specifications are automatically translated into an OWL ontology.

- OWL supports representing information in a structured way.

- OWL provides also support for querying the information.

- Information can be retrieved from the ontology resorting to the SPARQL query language.

The SCARP Approach - OWL

- Example of RUS to OWL mapping:

```
user selects the login
```

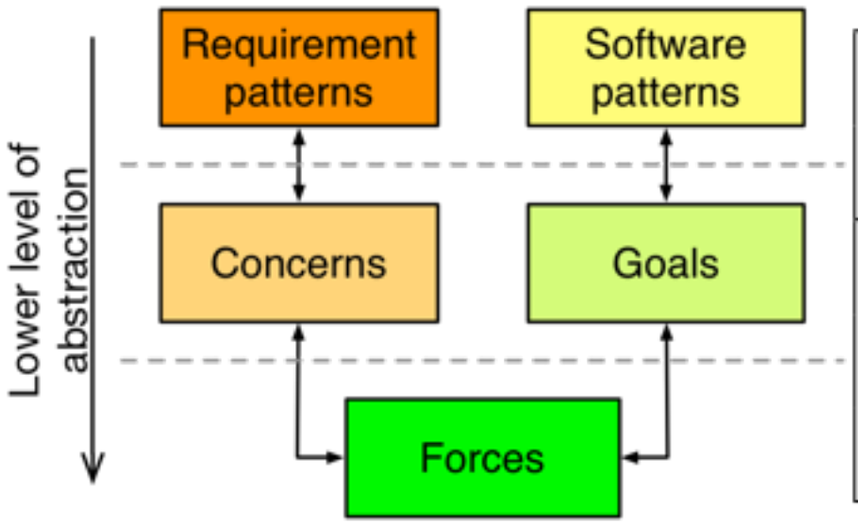
```
Individual: j.0:system

    Types:
        j.0:Actor

    Facts:
     j.0:selects  j.0:login
```

The SCARP Approach - Requirement Patterns

- **Requirement patterns** are common requirements found in different specifications. Examples:
  - *Has Shopping Cart*, *Has Catalog, Has Account.*
- Represent well known solutions for a given problem, within a certain context.
- The Use Cases Query Language (uQL) was developed to perform pattern inference.
- Inferring patterns consists in checking whether the condition is valid in the ontology.

The SCARP Approach - Requirement Patterns

- The requirement pattern to software pattern transition process is based on the analysis of patterns properties.

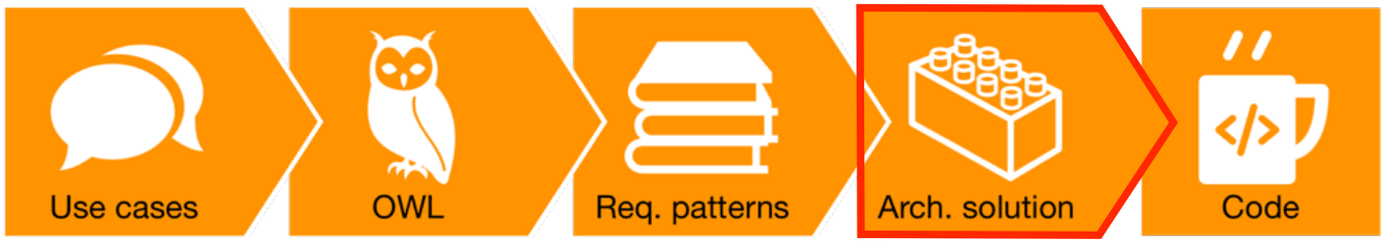


		Goals		
		Delegate	Edit	Process
Concerns	Processable	(+0, -1)	(+0, -1)	(+2, -1)
	Editable	(+0, -0)	(+2, -0)	(+0, -0)
	Manageable	(+2, -0)	(+1, -0)	(+1, -0)

The SCARP Approach - Requirement Patterns

Requirement pattern	Concern	Force	Goal	Software Patterns
Has Shopping Cart	Manageable	+ Separation	Edit	Memento
		+ Separation	Process	Command
		+ Separation	Delegate	**Proxy**
		+ Abstraction		
	Processable	+ Computability	Process	**Command**
		+ Indirection		
		- Separation	Edit	Memento
		- Separation	Process	Command
		- Separation	Delegate	Proxy
	Editable	+ Flexibility	Edit	**Memento**
		+ Versioning		

The SCARP Approach - Architectural Solutions



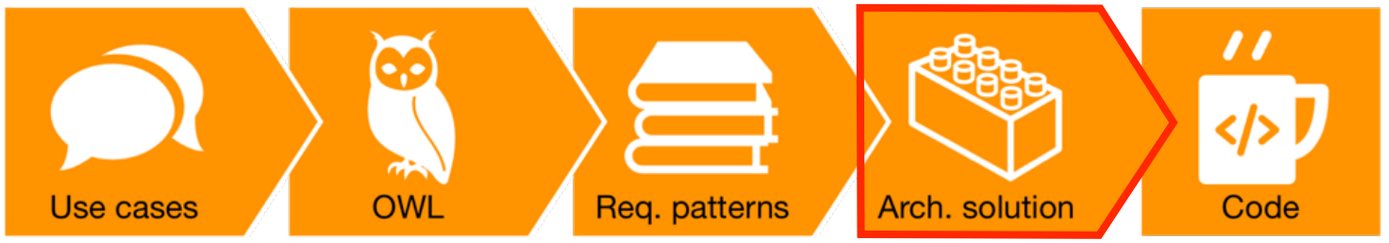
- An API-like format was defined to represent **software patterns** in a systematic and operationalizable way.

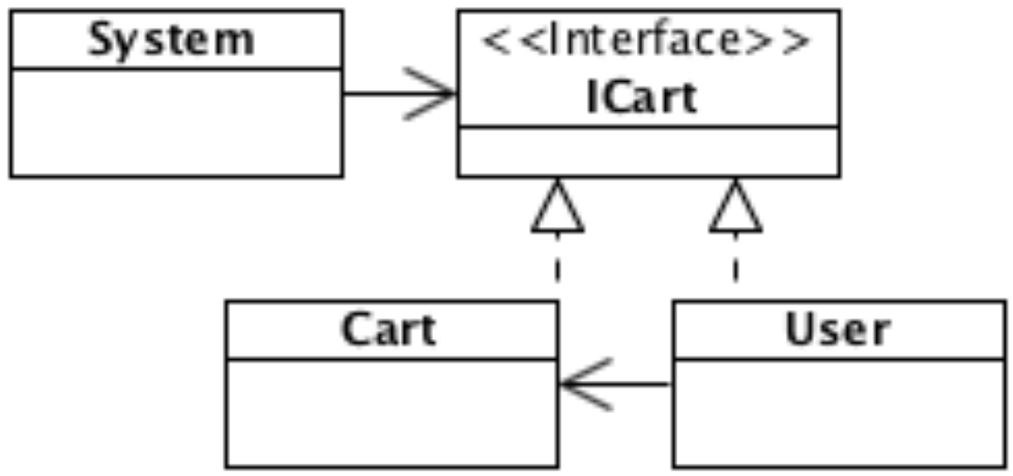
```
/**
* @intent Provide a surrogate or placeholder for another object to control access to it.
* @param client Triggers the request.
* @param subject Defines the common interface for RealSubject and Proxy.
* @param realSubject Defines the real object that the proxy represents.
* @param proxy Entry point to handle the request.
*/
proxy(client , subject , realSubject , proxy);
```

- The inferred software patterns are instantiated.

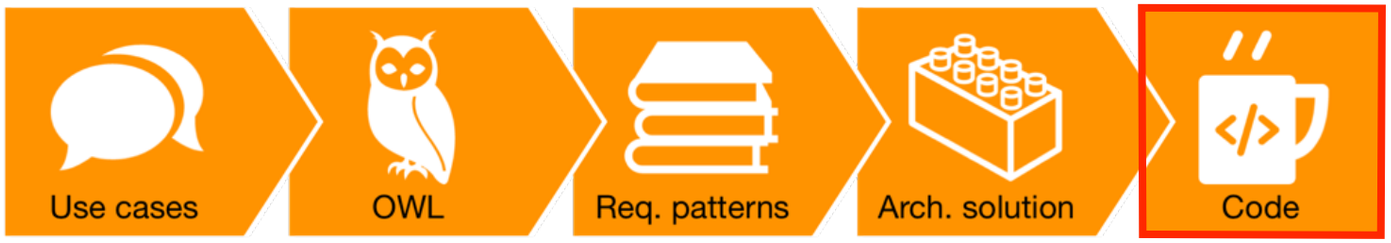
- The software pattern instances are combined in order to create an architectural solution.

The SCARP Approach - Architectural Solutions





The SCARP Approach - Code



- The produced outputs are architectural models.

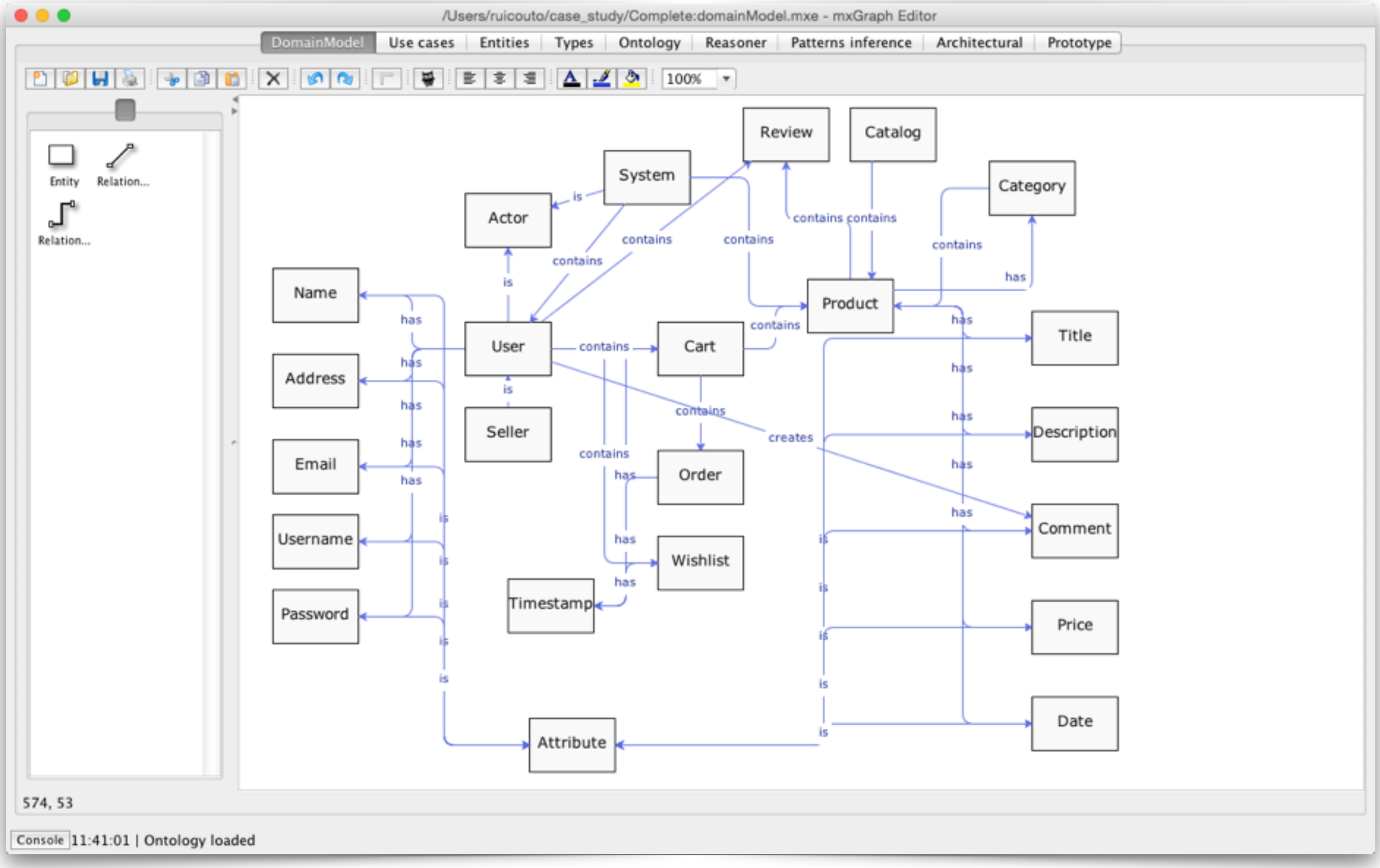
- Models are represented in XMI.

- The XMI interchangeable format supports interoperability, enabling the possibility to produce, for instance:

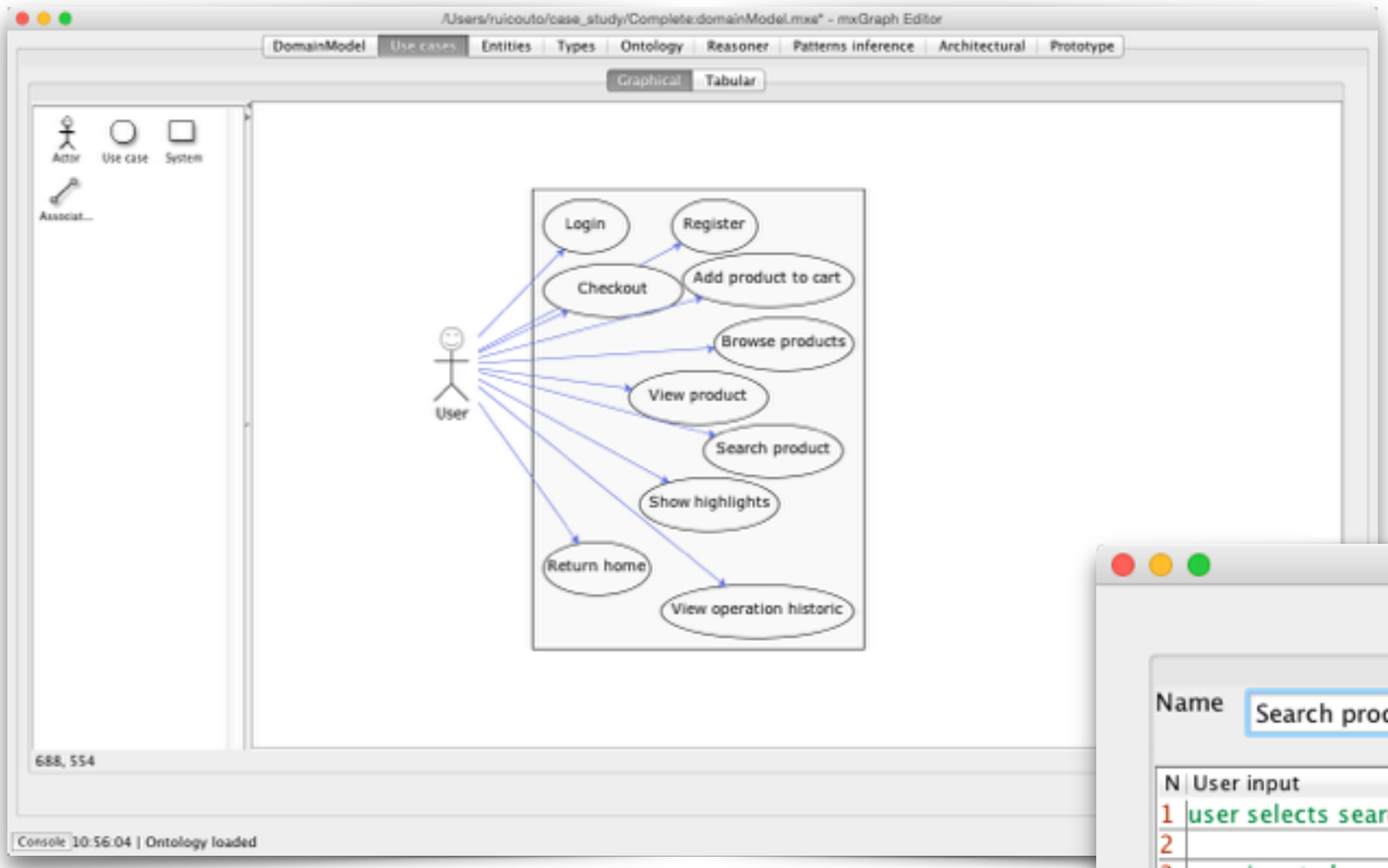
  - Source code

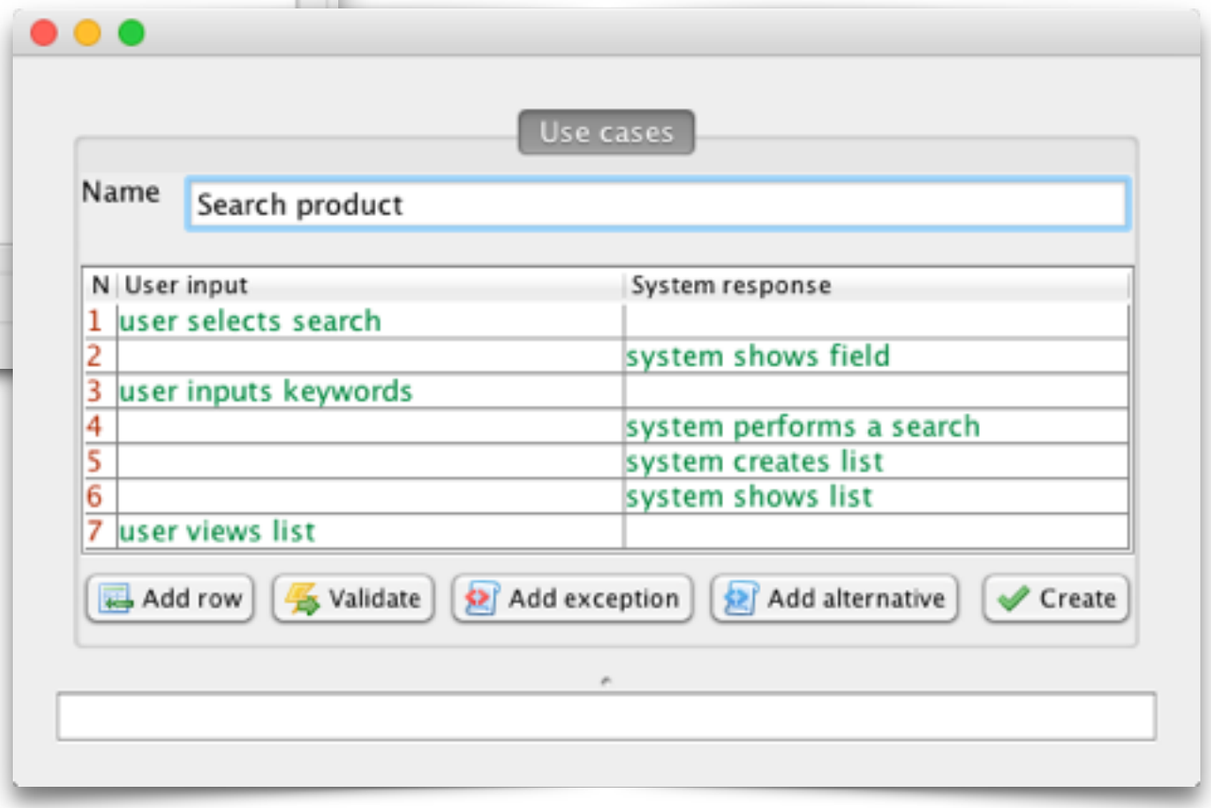
  - User interface prototypes

uCat tool

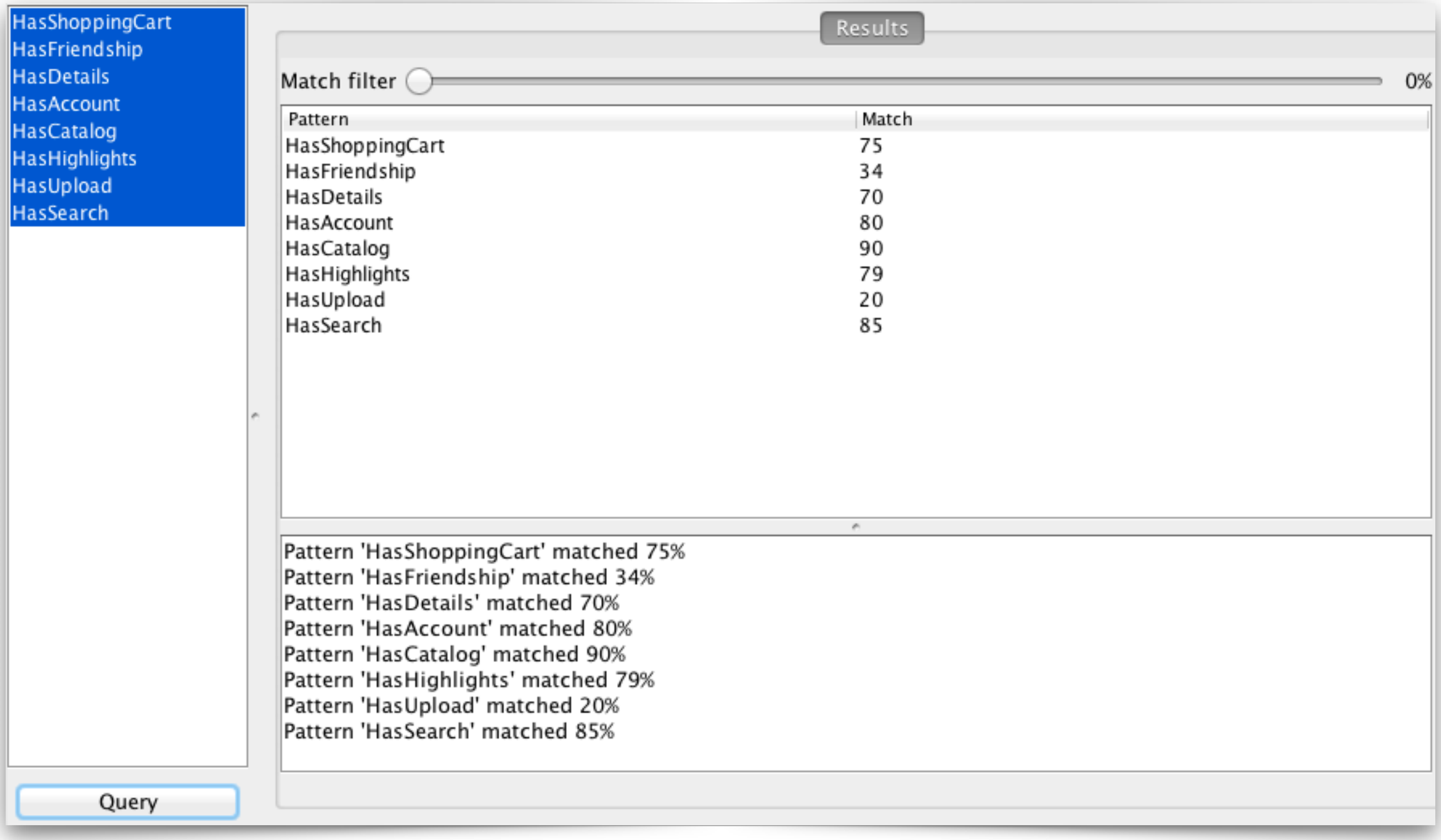

/Users/ruicouto/case_study/Complete:domainModel.mxe - mxGraph Editor
DomainModel
Use cases
Entities
Types
Ontology
Reasoner
Patterns inference
Architectural
Prototype
100%
Entity
Relation...
Relation...
Review
Catalog
System
Category
Actor
is
contains
contains
contains
contains
contains
contains
has
Name
Product
has
Title
User
contains
Cart
contains
has
Address
has
has
Seller
is
contains
Description
has
Email
has
has
creates
Order
has
has
Comment
Username
is
has
Wishlist
is
Timestamp
has
Password
is
Price
is
is
Date
is
Attribute
574, 53
Console
11:41:01 | Ontology loaded

uCat tool

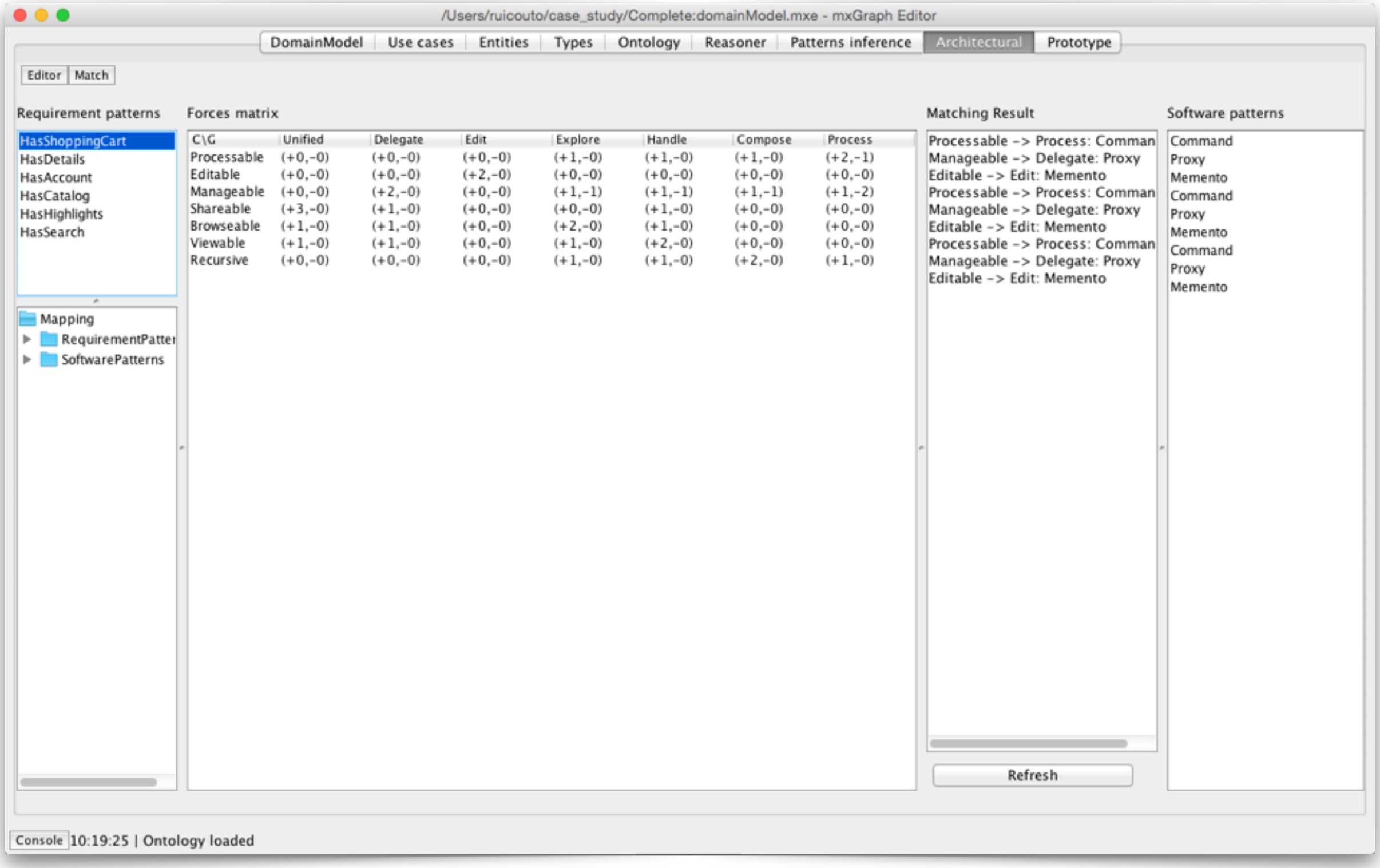

/Users/ruicouto/case_study/Complete:domainModel.mxa* - mxGraph Editor
DomainModel
Use cases
Entities
Types
Ontology
Reasoner
Patterns inference
Architectural
Prototype
Graphical
Tabular
Actor
Use case
System
Associat...
Login
Register
Checkout
Add product to cart
Browse products
View product
Search product
Show highlights
Return home
View operation historic
User
688, 554
Console 10:56:04 | Ontology loaded


Use cases
Name
Search product
N
User input
System response
1
user selects search
2
system shows field
3
user inputs keywords
4
system performs a search
5
system creates list
6
system shows list
7
user views list
Add row
Validate
Add exception
Add alternative
Create

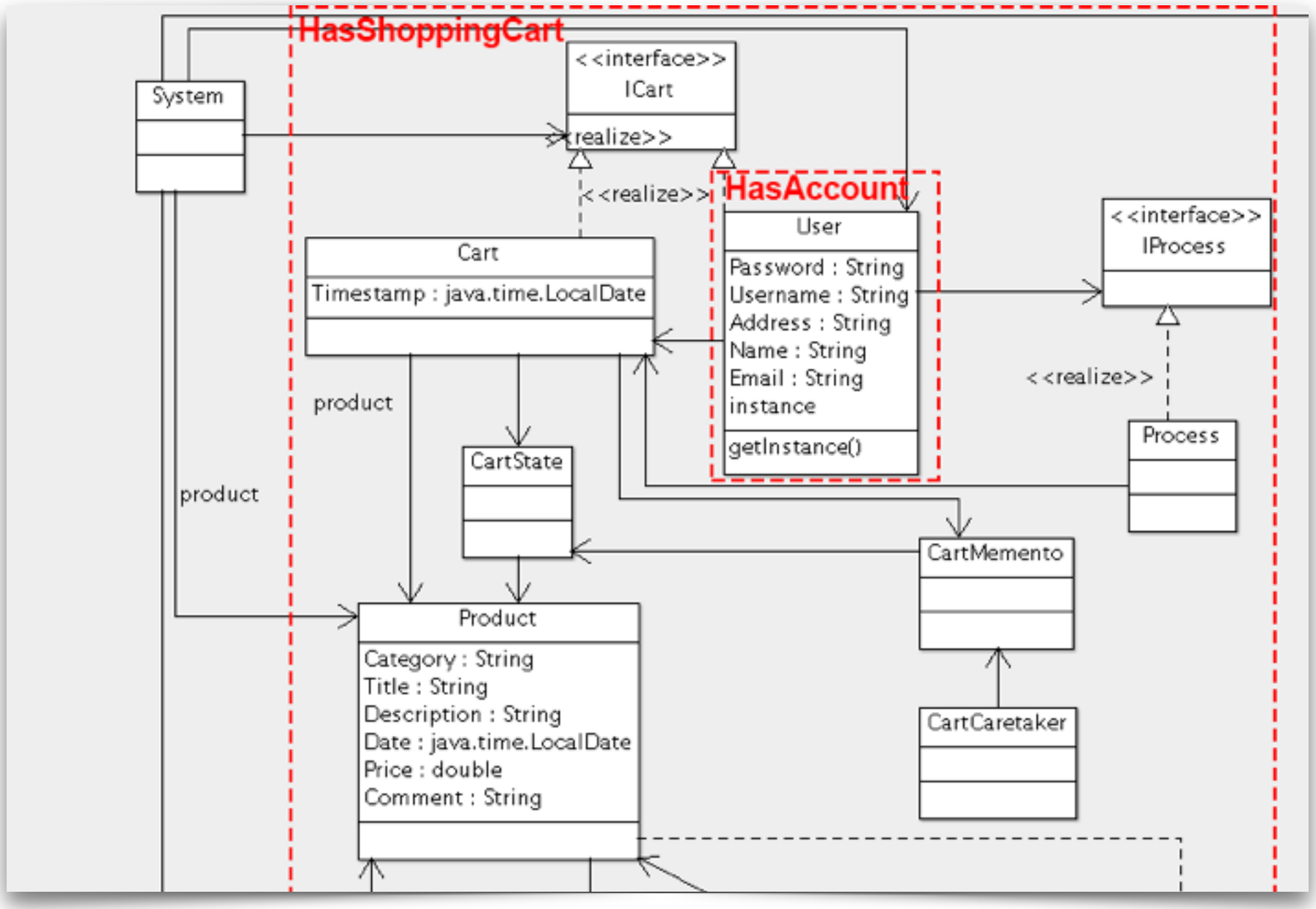
uCat tool


HasShoppingCart
HasFriendship
HasDetails
HasAccount
HasCatalog
HasHighlights
HasUpload
HasSearch
Query
Results
Match filter
0%
Pattern
Match
HasShoppingCart 75
HasFriendship 34
HasDetails 70
HasAccount 80
HasCatalog 90
HasHighlights 79
HasUpload 20
HasSearch 85
Pattern 'HasShoppingCart' matched 75%
Pattern 'HasFriendship' matched 34%
Pattern 'HasDetails' matched 70%
Pattern 'HasAccount' matched 80%
Pattern 'HasCatalog' matched 90%
Pattern 'HasHighlights' matched 79%
Pattern 'HasUpload' matched 20%
Pattern 'HasSearch' matched 85%

uCat tool


/Users/ruicouto/case_study/Complete:domainModel.mxe - mxGraph Editor
DomainModel
Use cases
Entities
Types
Ontology
Reasoner
Patterns inference
Architectural
Prototype
Editor
Match
Requirement patterns
HasShoppingCart
HasDetails
HasAccount
HasCatalog
HasHighlights
HasSearch
Mapping
RequirementPatter
SoftwarePatterns
Forces matrix
C\G Unified Delegate Edit Explore Handle Compose Process
Processable (+0,-0) (+0,-0) (+0,-0) (+1,-0) (+1,-0) (+1,-0) (+2,-1)
Editable (+0,-0) (+0,-0) (+2,-0) (+0,-0) (+0,-0) (+0,-0) (+0,-0)
Manageable (+0,-0) (+2,-0) (+0,-0) (+1,-1) (+1,-1) (+1,-1) (+1,-2)
Shareable (+3,-0) (+1,-0) (+0,-0) (+0,-0) (+1,-0) (+0,-0) (+0,-0)
Browseable (+1,-0) (+1,-0) (+0,-0) (+2,-0) (+1,-0) (+0,-0) (+0,-0)
Viewable (+1,-0) (+1,-0) (+0,-0) (+1,-0) (+2,-0) (+0,-0) (+0,-0)
Recursive (+0,-0) (+0,-0) (+0,-0) (+1,-0) (+1,-0) (+2,-0) (+1,-0)
Matching Result
Processable -> Process: Comman
Manageable -> Delegate: Proxy
Editable -> Edit: Memento
Processable -> Process: Comman
Manageable -> Delegate: Proxy
Editable -> Edit: Memento
Processable -> Process: Comman
Manageable -> Delegate: Proxy
Editable -> Edit: Memento
Refresh
Software patterns
Command
Proxy
Memento
Command
Proxy
Memento
Command
Proxy
Memento
Console
10:19:25 | Ontology loaded

uCat tool


HasShoppingCart
System
<<interface>>
ICart
realize>>
<<realize>>
HasAccount
User
Password : String
Username : String
Address : String
Name : String
Email : String
instance
getInstance()
<<interface>>
IProcess
<<realize>>
Process
Cart
Timestamp : java.time.LocalDate
product
product
product
CartState
CartMemento
CartCaretaker
Product
Category : String
Title : String
Description : String
Date : java.time.LocalDate
Price : double
Comment : String

Validation

- Two studies were performed in order to validate the approach.
- In the first study the expressiveness of the language, suitability of uCat, and usability were addressed.
  - The language expressiveness was successfully validated.
  - The tool had a good usability and SUS score (score 74, grade B).
- In the second study was validated the requirements patterns inference mechanism.
  - Patterns were successfully inferred from specifications.

Conclusions and Future Work

- Currently, the MDA starts the development process with architectural models.

- Since requirements affect the software solutions, the process should start in a earlier phase.

- The transition from requirement to architectural is currently manual.

- The SCARP approach enables the possibility to automate the transformation of software models.

Conclusions and Future Work

- The uCat tool successfully supports and automates the approach.
- Two performed validation studies support the viability of both SCARP and uCat.
- The SCARP process can be extended to address behavioral aspects, as well as traceability.
- RUS can be improved, taking more advantage from OWL.
- Other domains can be explored.
- The application of formal verification techniques can be explored.

Contributions

- A controlled natural language to formalize requirements, and automatic transformation into OWL.
- A software pattern inference mechanism.
- A systematic requirement to software pattern transition process.
- A tool supporting the SCARP process.

Publications

- Rui Couto et al. MapIt: A model based pattern recovery tool. In MOMPES 2012.
- Rui Couto et al. A patterns based reverse engineering approach for java source code. In SEW 2012.
- Rui Couto et al. Application of ontologies in identifying requirements patterns in use cases. In FESCA 2014.
- Rui Couto et al. A study on the viability of formalizing use cases. In QUATIC 2014.
- Rui Couto et al. The Modelery: A collaborative web based repository. In ICCSA 2014.
- Rui Couto et al. The Modelery: a model-based software development repository. In IJWIS 2015.
- Rui Couto et al. Validating an approach to formalize use cases with ontologies. In FESCA 2016.
- Marina Machado et al. Modus: uma metodologia de prototipagem de interfaces baseada em modelos. In INFORUM 2015.

Pattern Based Software Development

Rui Couto
António Nestor Ribeiro
José Creissac Campos