Requirement Traceability: A Model-Based Approach

Omar Badreddin - Arnon Sturm - Timothy C. Lethbridge
Omar.Badreddin@nau.edu - sturm@bgu.ac.il - tcl@eecs.uottawa.ca

Traceability in Software Engineering

- Tractability is about:
  - The ability to <u>interrelate</u> any uniquely identifiable artifact to any other;
  - To <u>maintain</u> links over time; and
  - To <u>use</u> the resulting network to answer questions of both the software product and its development process

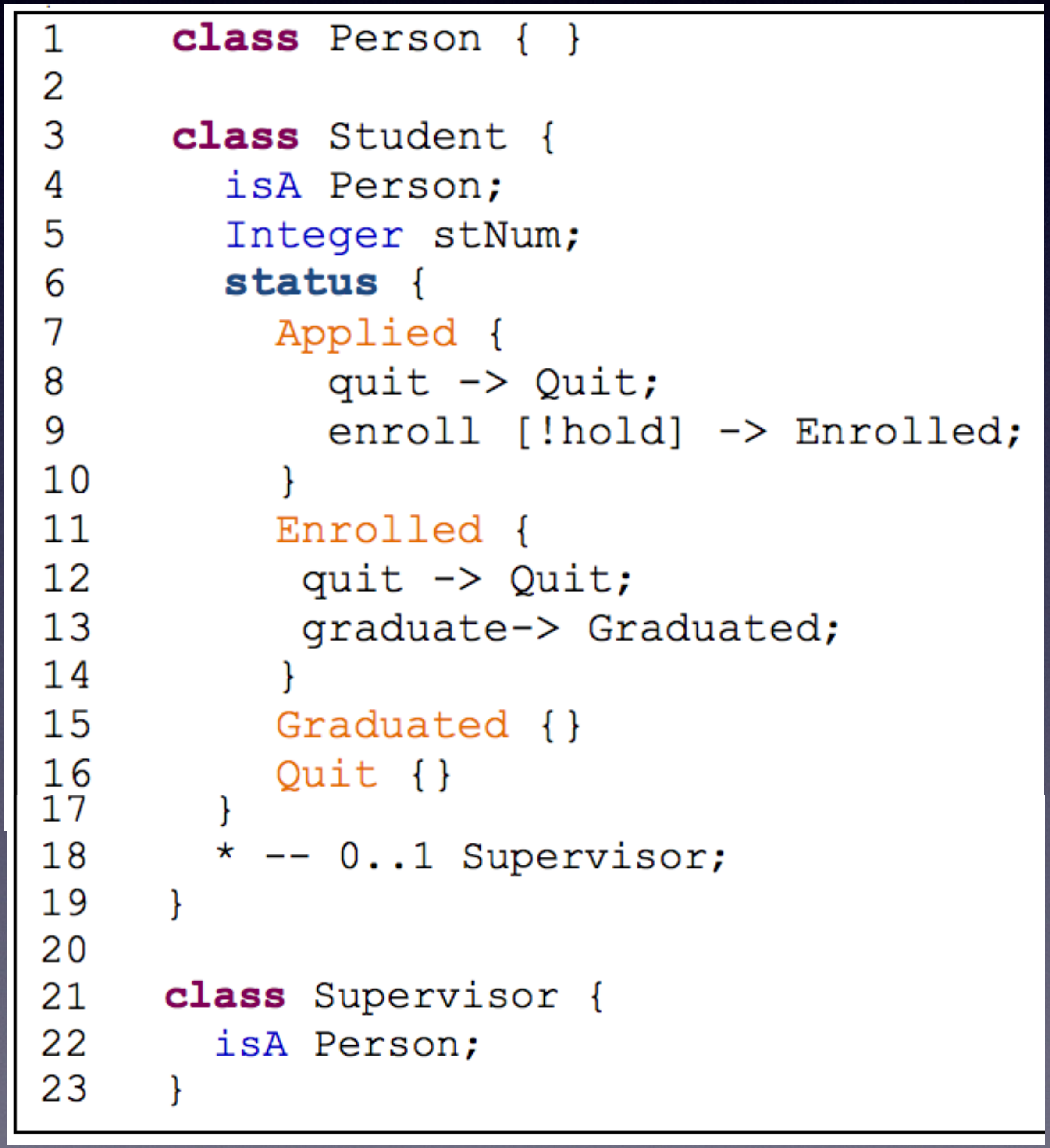
Why Trace Requirements

- Verify Coverage
- Avoid Redundancy
- Assess Impact of Changes
- Requirements Traces must be
    - maintained as the system evolves
    - Available at run time (maybe?)
    - Accessible at variable levels of abstractions

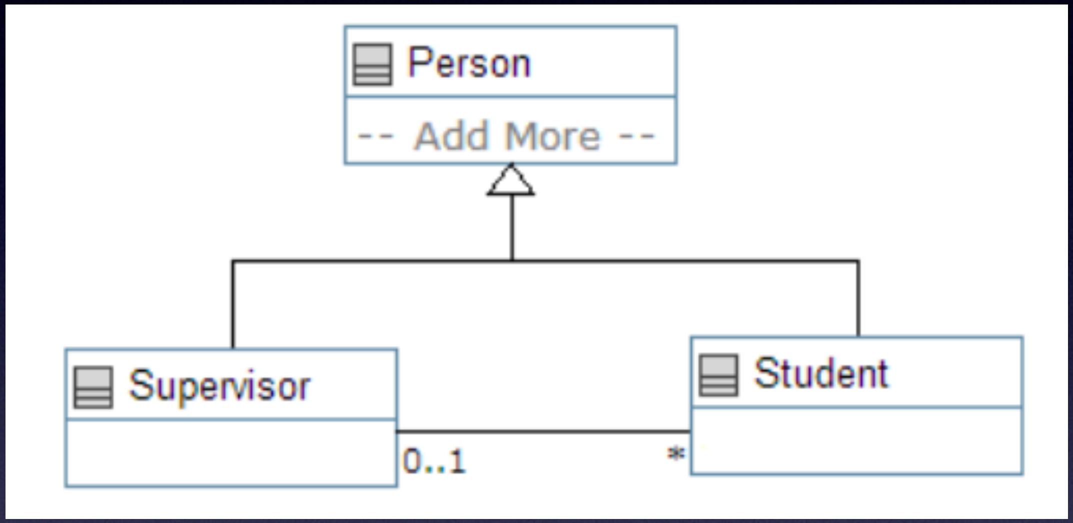
Current Practice

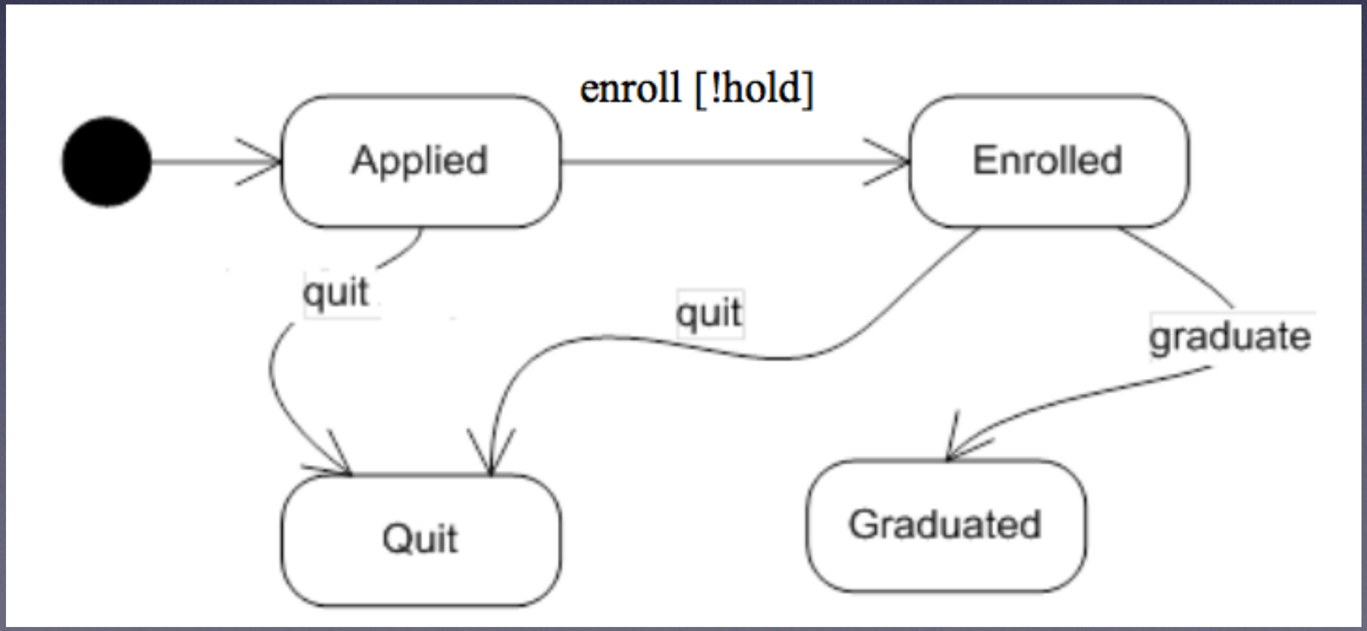
- Requirements activities are separate from development activities
- Links between requirements and development artifacts must be maintained
- MDA Approach
  - Focus on system entities and behaviour
  - Less emphasis on system goals, actors, and non functional requirements.

Model Oriented Programming

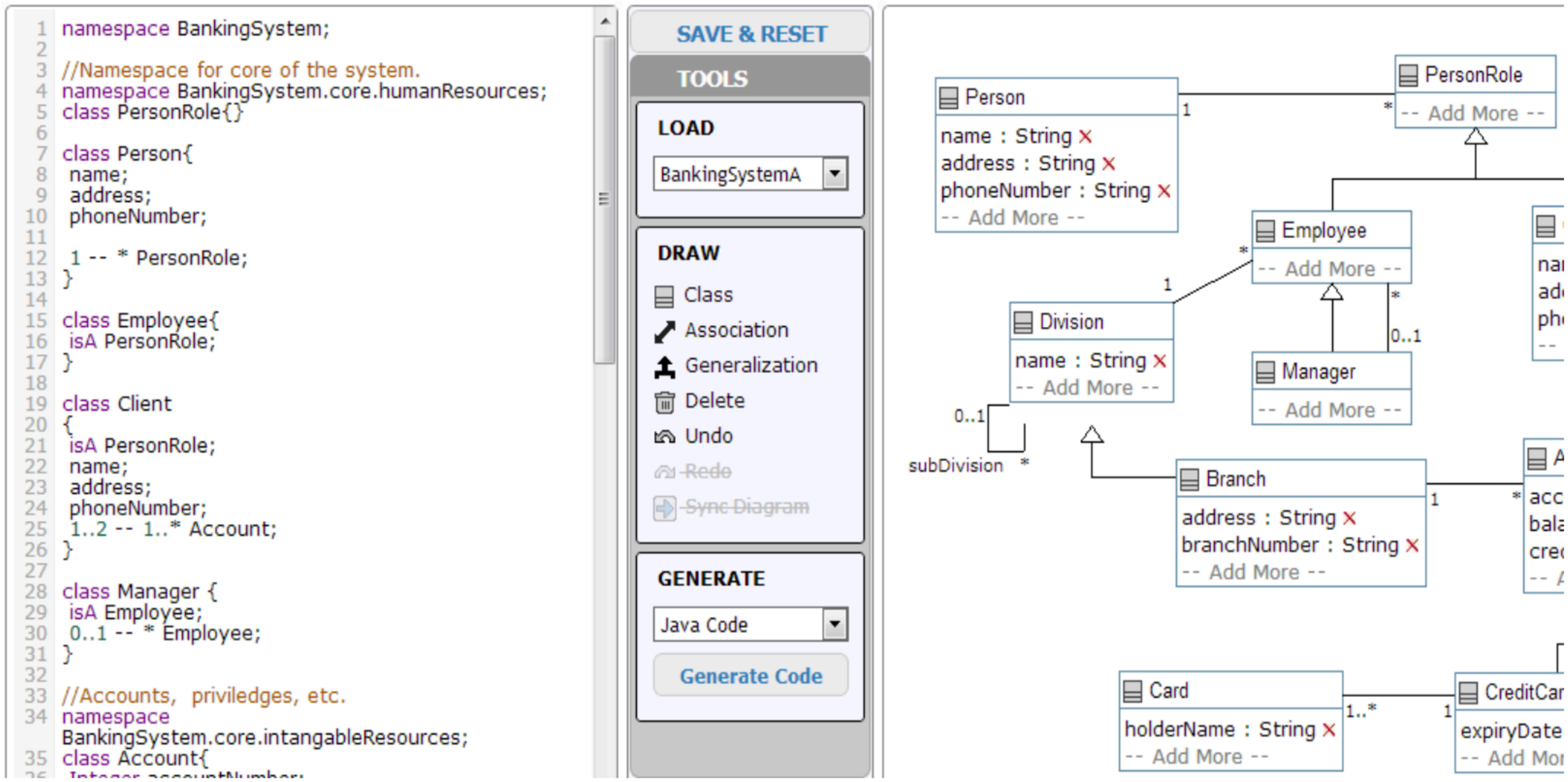


```
class Person { }

class Student {
  isA Person;
  Integer stNum;
  status {
    Applied {
      quit -> Quit;
      enroll [!hold] -> Enrolled;
    }
    Enrolled {
     quit -> Quit;
     graduate-> Graduated;
    }
    Graduated {}
    Quit {}
  }
  * -- 0..1 Supervisor;
}

class Supervisor {
  isA Person;
}
```





Umple Online
www.try.umple.org



```
namespace BankingSystem;

//Namespace for core of the system.
namespace BankingSystem.core.humanResources;
class PersonRole{}

class Person{
 name;
 address;
 phoneNumber;

 1 -- * PersonRole;
}

class Employee{
 isA PersonRole;
}

class Client
{
 isA PersonRole;
 name;
 address;
 phoneNumber;
 1..2 -- 1..* Account;
}

class Manager {
 isA Employee;
 0..1 -- * Employee;
}

//Accounts, priviledges, etc.
namespace
BankingSystem.core.intangableResources;
class Account{
```

Overview of Research Direction

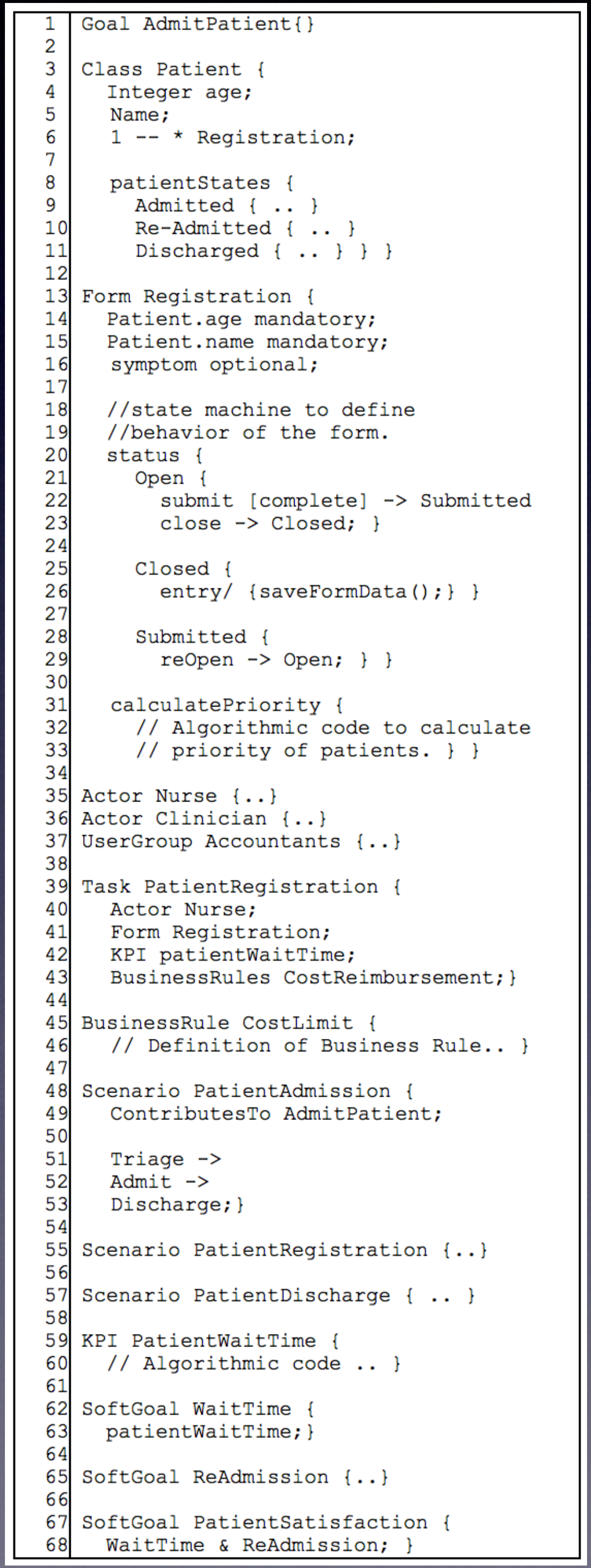
- Umple, a Model Oriented Programming Platform
  - Enhance OO code with modeling abstractions
  - Associations, State Machine, Model Based Tracing, OCL like constraints.
  - Visual and Textual views are automatically synchronized
- No need to edit the generated code

Adding Key Requirements Entities

- We propose to incorporate textual representation of key requirements entities into the ”Model Oriented” code.
- Thus, one can interplay between requirements, models, and code.
- As a results, we eliminate or reduce the need for creating or maintaining requirements links.

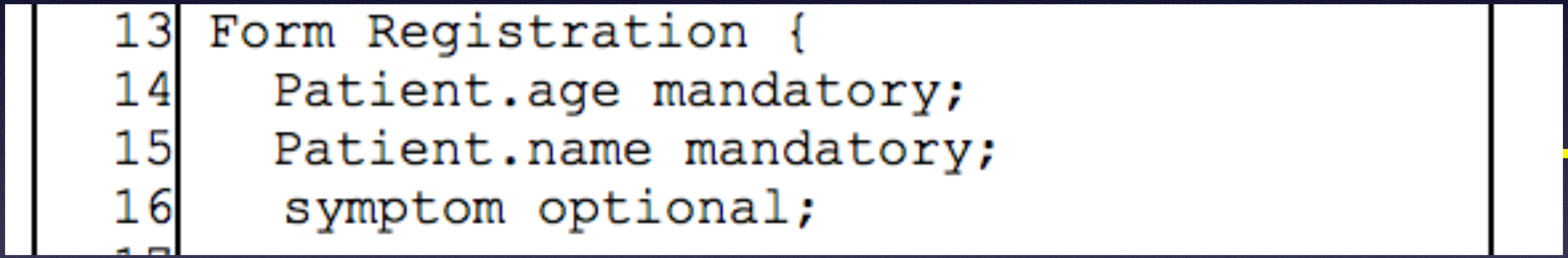
Requirement-Oriented Model and Programming Language (ROMPL)

- Language components
  - OO code
  - Modeling Abstractions (state machines, Associations, etc..)
  - Requirements Entities
    - Goals, KPIs, Business Rules, ..

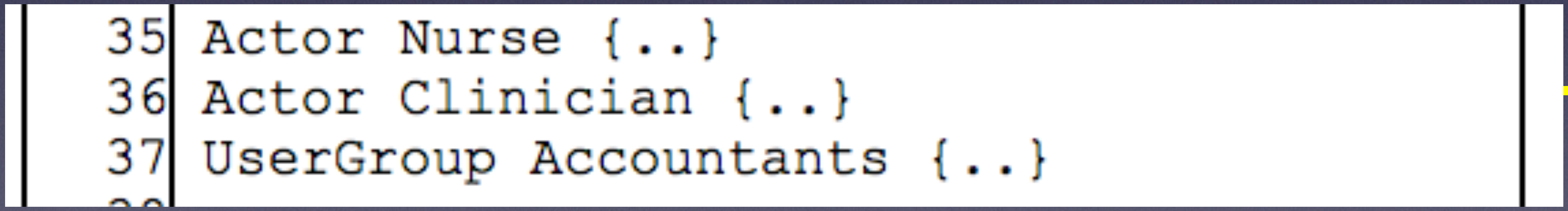


```
Goal AdmitPatient{}

Class Patient {
  Integer age;
  Name;
  1 -- * Registration;

  patientStates {
    Admitted { .. }
    Re-Admitted { .. }
    Discharged { .. } } }

Form Registration {
  Patient.age mandatory;
  Patient.name mandatory;
  symptom optional;

  //state machine to define
  //behavior of the form.
  status {
    Open {
      submit [complete] -> Submitted
      close -> Closed; }

    Closed {
      entry/ {saveFormData();} }

    Submitted {
      reOpen -> Open; } }

  calculatePriority {
    // Algorithmic code to calculate
    // priority of patients. } }

Actor Nurse {..}
Actor Clinician {..}
UserGroup Accountants {..}

Task PatientRegistration {
  Actor Nurse;
  Form Registration;
  KPI patientWaitTime;
  BusinessRules CostReimbursement;}

BusinessRule CostLimit {
  // Definition of Business Rule.. }

Scenario PatientAdmission {
  ContributesTo AdmitPatient;

  Triage ->
  Admit ->
  Discharge;}

Scenario PatientRegistration {..}

Scenario PatientDischarge { .. }

KPI PatientWaitTime {
  // Algorithmic code .. }

SoftGoal WaitTime {
  patientWaitTime;}

SoftGoal ReAdmission {..}

SoftGoal PatientSatisfaction {
  WaitTime & ReAdmission; }
```

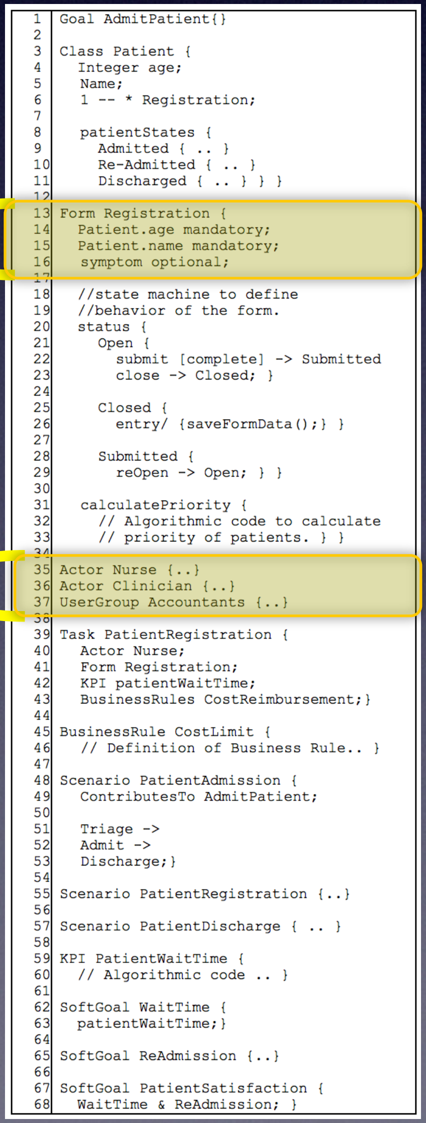
Forms & Users



```
Form Registration {
  Patient.age mandatory;
  Patient.name mandatory;
   symptom optional;
```

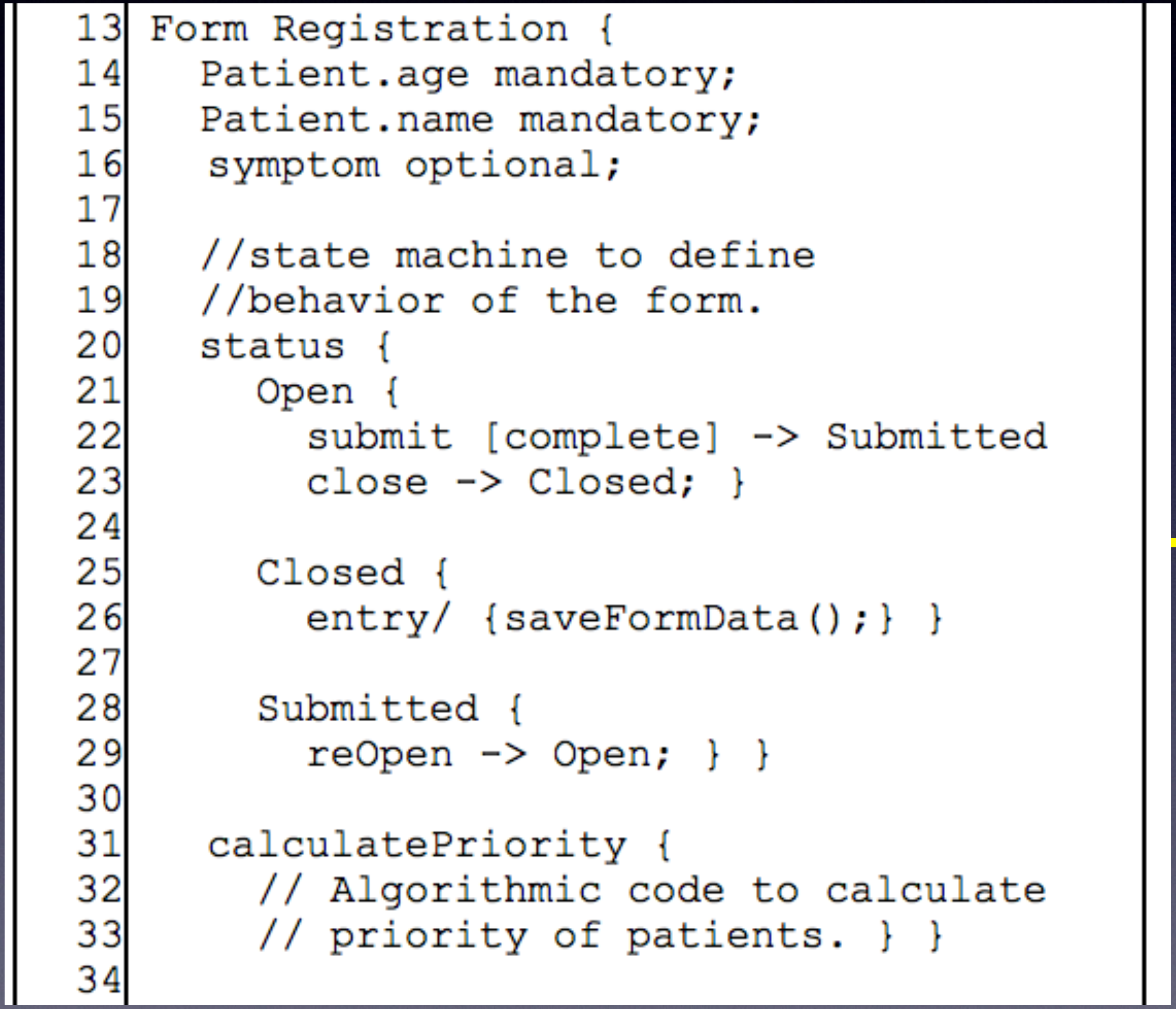


```
Actor Nurse {..}
Actor Clinician {..}
UserGroup Accountants {..}
```

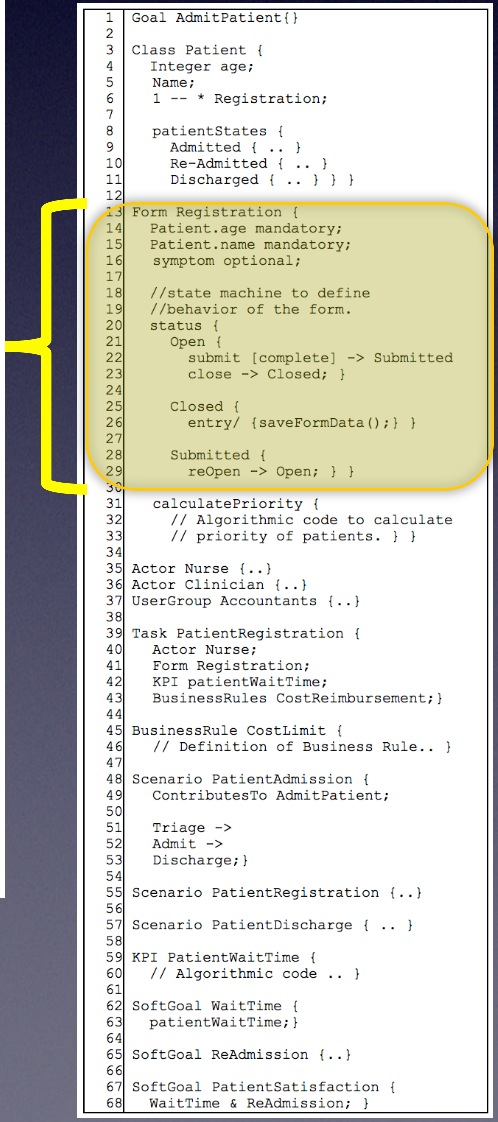


```
Goal AdmitPatient{}

Class Patient {
  Integer age;
   Name;
   1 -- * Registration;

   patientStates {
     Admitted { .. }
     Re-Admitted { .. }
     Discharged { .. } } }

Form Registration {
  Patient.age mandatory;
  Patient.name mandatory;
   symptom optional;

  //state machine to define
  //behavior of the form.
  status {
    Open {
      submit [complete] -> Submitted
      close -> Closed; }

    Closed {
      entry/ {saveFormData();} }

    Submitted {
      reOpen -> Open; } }

   calculatePriority {
     // Algorithmic code to calculate
     // priority of patients. } }

Actor Nurse {..}
Actor Clinician {..}
UserGroup Accountants {..}

Task PatientRegistration {
   Actor Nurse;
   Form Registration;
   KPI patientWaitTime;
   BusinessRules CostReimbursement;}

BusinessRule CostLimit {
   // Definition of Business Rule.. }

Scenario PatientAdmission {
   ContributesTo AdmitPatient;

   Triage ->
   Admit ->
   Discharge;}

Scenario PatientRegistration {..}

Scenario PatientDischarge { .. }

KPI PatientWaitTime {
  // Algorithmic code .. }

SoftGoal WaitTime {
  patientWaitTime;}

SoftGoal ReAdmission {..}

SoftGoal PatientSatisfaction {
  WaitTime & ReAdmission; }
```

State Machine Modeling

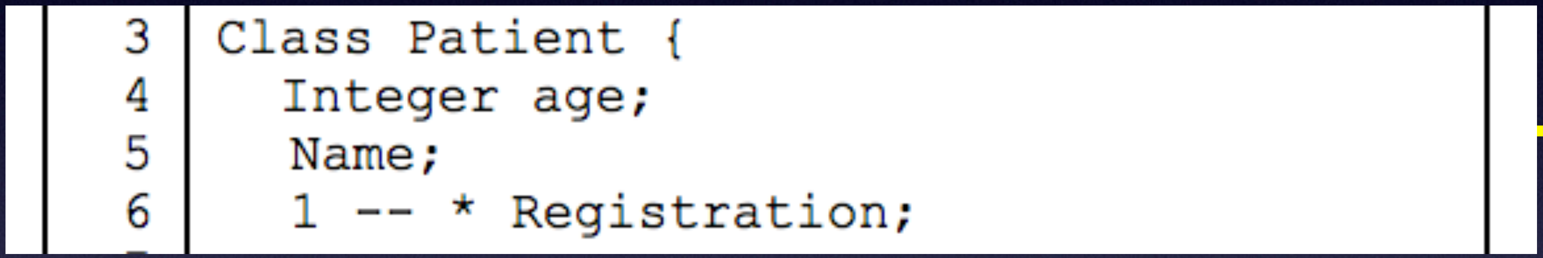


```
Form Registration {
  Patient.age mandatory;
  Patient.name mandatory;
   symptom optional;

  //state machine to define
  //behavior of the form.
  status {
    Open {
      submit [complete] -> Submitted
      close -> Closed; }

    Closed {
      entry/ {saveFormData();} }

    Submitted {
      reOpen -> Open; } }

   calculatePriority {
     // Algorithmic code to calculate
     // priority of patients. } }
```

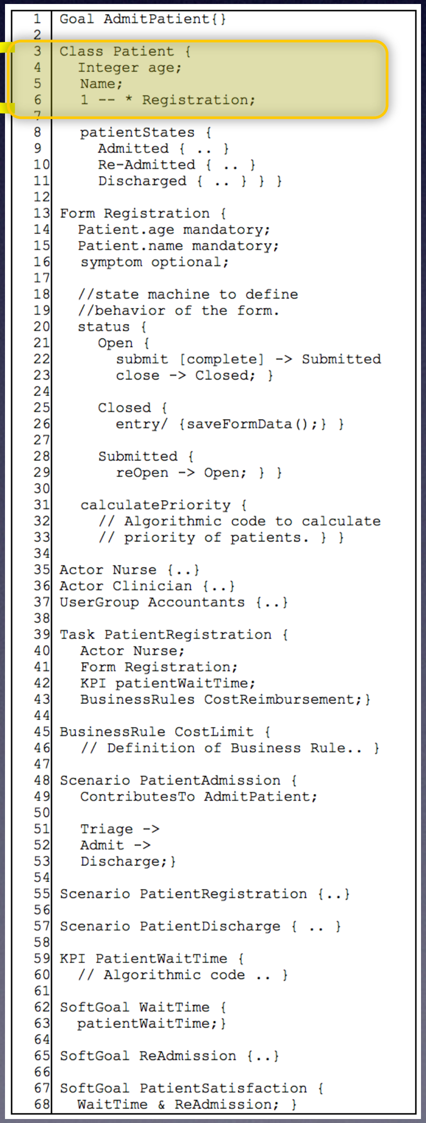


```
Goal AdmitPatient{}

Class Patient {
  Integer age;
   Name;
   1 -- * Registration;

   patientStates {
     Admitted { .. }
     Re-Admitted { .. }
     Discharged { .. } } }

Form Registration {
  Patient.age mandatory;
  Patient.name mandatory;
   symptom optional;

  //state machine to define
  //behavior of the form.
  status {
    Open {
      submit [complete] -> Submitted
      close -> Closed; }

    Closed {
      entry/ {saveFormData();} }

    Submitted {
      reOpen -> Open; } }

   calculatePriority {
     // Algorithmic code to calculate
     // priority of patients. } }

Actor Nurse {..}
Actor Clinician {..}
UserGroup Accountants {..}

Task PatientRegistration {
   Actor Nurse;
   Form Registration;
   KPI patientWaitTime;
   BusinessRules CostReimbursement;}

BusinessRule CostLimit {
   // Definition of Business Rule.. }

Scenario PatientAdmission {
   ContributesTo AdmitPatient;

   Triage ->
   Admit ->
   Discharge;}

Scenario PatientRegistration {..}

Scenario PatientDischarge { .. }

KPI PatientWaitTime {
  // Algorithmic code .. }

SoftGoal WaitTime {
  patientWaitTime;}

SoftGoal ReAdmission {..}

SoftGoal PatientSatisfaction {
  WaitTime & ReAdmission; }
```

UML Attributes and Associations



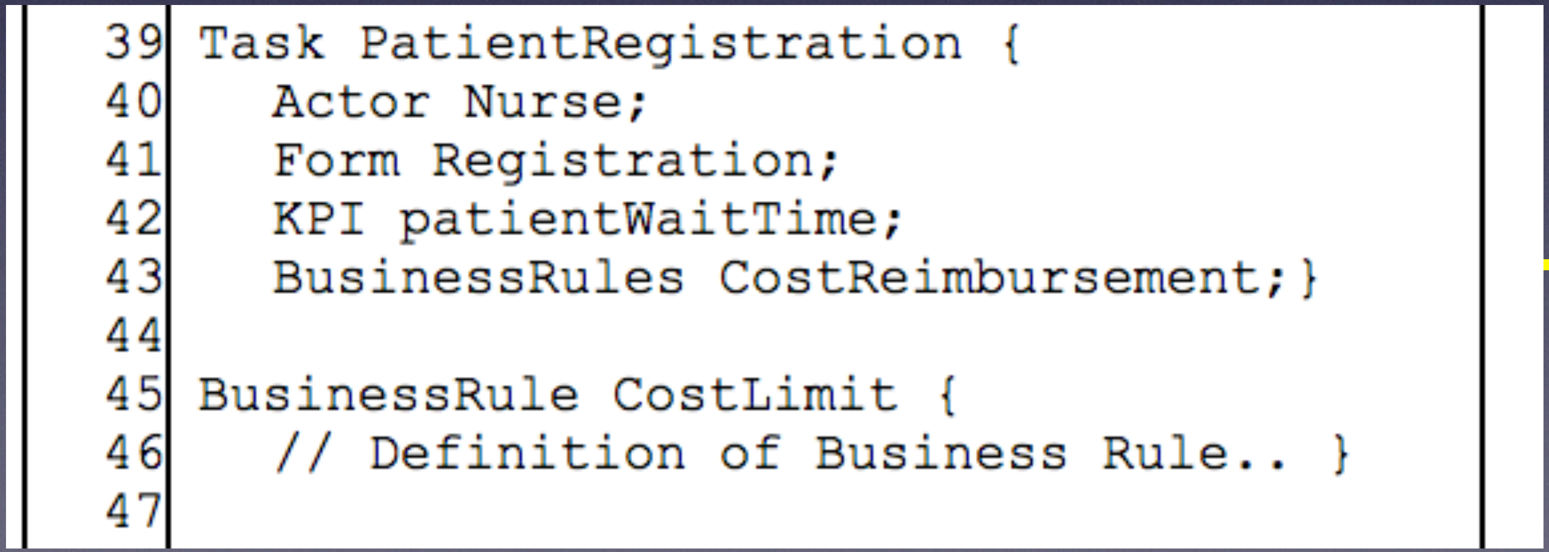
```
Class Patient {
  Integer age;
  Name;
  1 -- * Registration;
```



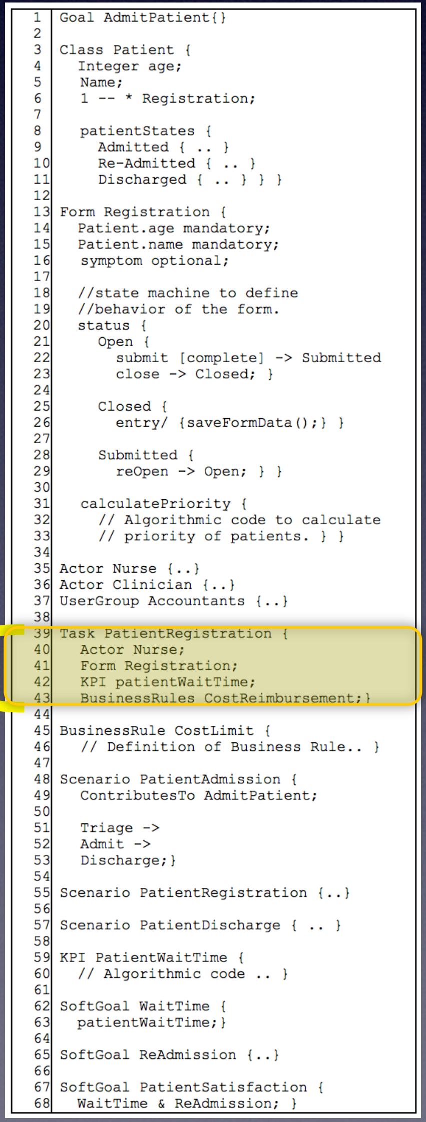
```
Goal AdmitPatient{}

Class Patient {
  Integer age;
  Name;
  1 -- * Registration;

  patientStates {
    Admitted { .. }
    Re-Admitted { .. }
    Discharged { .. } } }

Form Registration {
  Patient.age mandatory;
  Patient.name mandatory;
  symptom optional;

  //state machine to define
  //behavior of the form.
  status {
    Open {
      submit [complete] -> Submitted
      close -> Closed; }

    Closed {
      entry/ {saveFormData();} }

    Submitted {
      reOpen -> Open; } }

  calculatePriority {
    // Algorithmic code to calculate
    // priority of patients. } }

Actor Nurse {..}
Actor Clinician {..}
UserGroup Accountants {..}

Task PatientRegistration {
  Actor Nurse;
  Form Registration;
  KPI patientWaitTime;
  BusinessRules CostReimbursement;}

BusinessRule CostLimit {
  // Definition of Business Rule.. }

Scenario PatientAdmission {
  ContributesTo AdmitPatient;

  Triage ->
  Admit ->
  Discharge;}

Scenario PatientRegistration {..}

Scenario PatientDischarge { .. }

KPI PatientWaitTime {
  // Algorithmic code .. }

SoftGoal WaitTime {
  patientWaitTime;}

SoftGoal ReAdmission {..}

SoftGoal PatientSatisfaction {
  WaitTime & ReAdmission; }
```

Tasks & Business Rules

- Performed by Actors
- May involve completing Forms
- Measured by KPIs
- Must conform to Business Rules



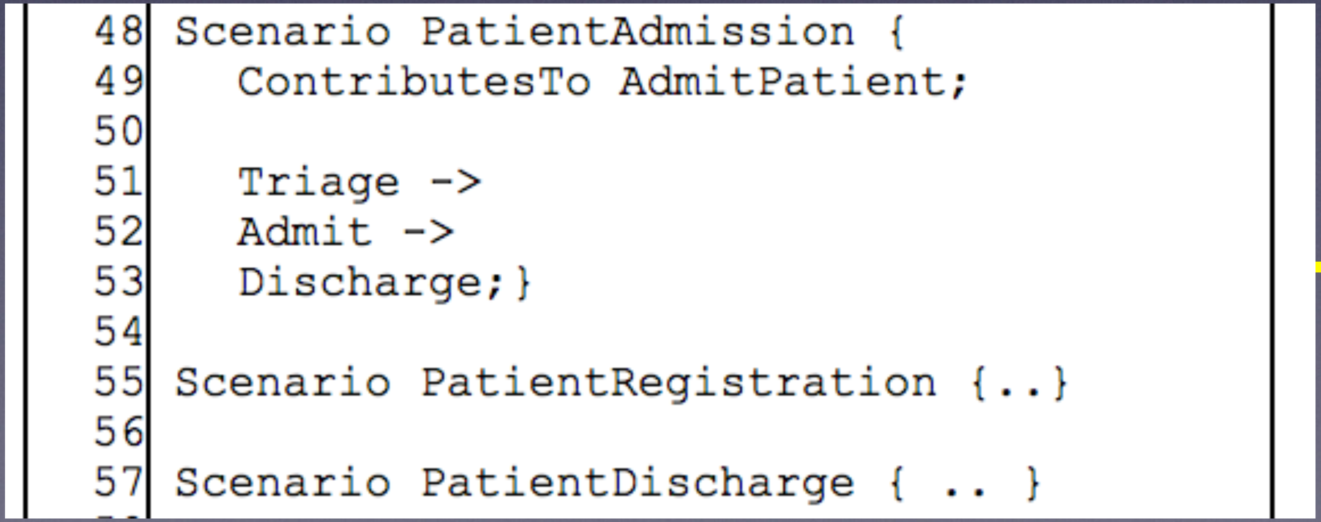
```
Task PatientRegistration {
  Actor Nurse;
  Form Registration;
  KPI patientWaitTime;
  BusinessRules CostReimbursement;}

BusinessRule CostLimit {
  // Definition of Business Rule.. }

```



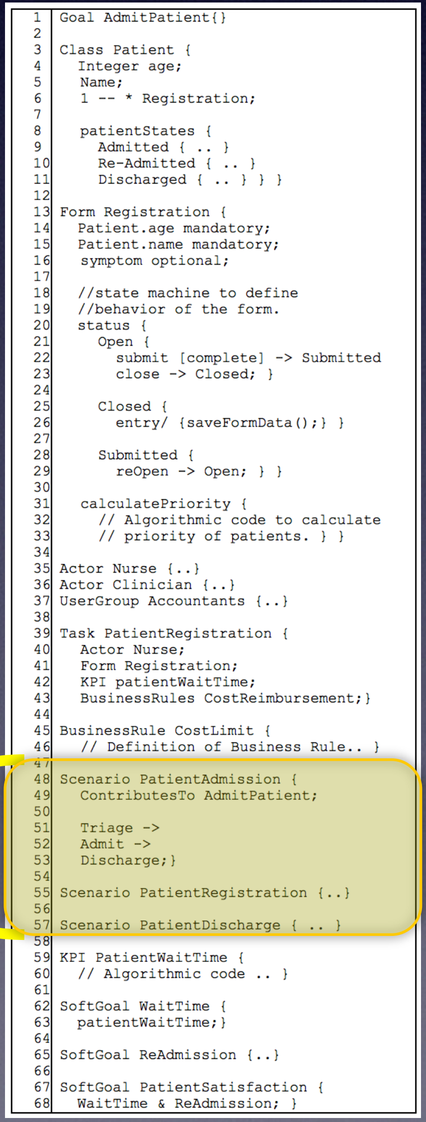
```
Goal AdmitPatient{}

Class Patient {
  Integer age;
  Name;
  1 -- * Registration;

  patientStates {
    Admitted { .. }
    Re-Admitted { .. }
    Discharged { .. } } }

Form Registration {
  Patient.age mandatory;
  Patient.name mandatory;
  symptom optional;

  //state machine to define
  //behavior of the form.
  status {
    Open {
      submit [complete] -> Submitted
      close -> Closed; }

    Closed {
      entry/ {saveFormData();} }

    Submitted {
      reOpen -> Open; } }

  calculatePriority {
    // Algorithmic code to calculate
    // priority of patients. } }

Actor Nurse {..}
Actor Clinician {..}
UserGroup Accountants {..}

Task PatientRegistration {
  Actor Nurse;
  Form Registration;
  KPI patientWaitTime;
  BusinessRules CostReimbursement;}

BusinessRule CostLimit {
  // Definition of Business Rule.. }

Scenario PatientAdmission {
  ContributesTo AdmitPatient;

  Triage ->
  Admit ->
  Discharge;}

Scenario PatientRegistration {..}

Scenario PatientDischarge { .. }

KPI PatientWaitTime {
  // Algorithmic code .. }

SoftGoal WaitTime {
  patientWaitTime;}

SoftGoal ReAdmission {..}

SoftGoal PatientSatisfaction {
  WaitTime & ReAdmission; }
```

Scenarios

- Scenarios are a sequence of tasks
- Support for Forks and Joins



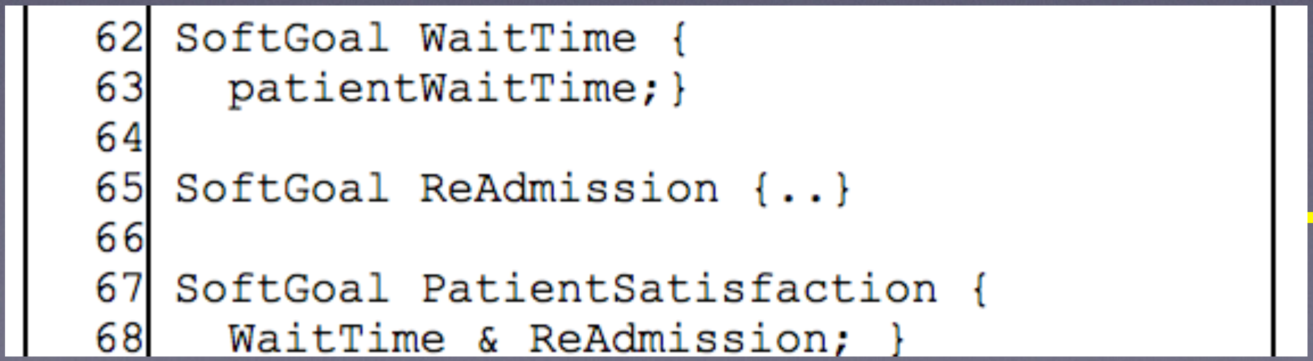
```
Scenario PatientAdmission {
  ContributesTo AdmitPatient;

  Triage ->
  Admit ->
  Discharge;}

Scenario PatientRegistration {..}

Scenario PatientDischarge { .. }
```



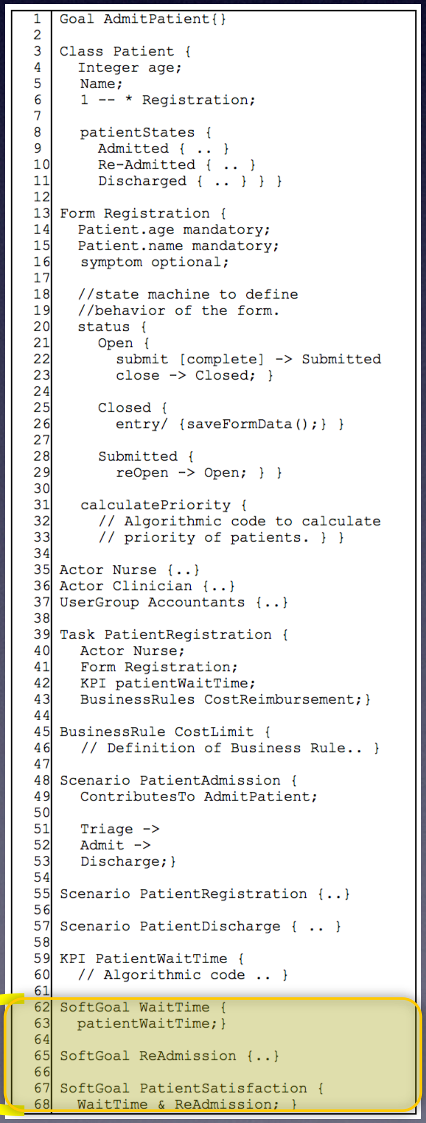
```
Goal AdmitPatient{}

Class Patient {
  Integer age;
  Name;
  1 -- * Registration;

  patientStates {
    Admitted { .. }
    Re-Admitted { .. }
    Discharged { .. } } }

Form Registration {
  Patient.age mandatory;
  Patient.name mandatory;
  symptom optional;

  //state machine to define
  //behavior of the form.
  status {
    Open {
      submit [complete] -> Submitted
      close -> Closed; }

    Closed {
      entry/ {saveFormData();} }

    Submitted {
      reOpen -> Open; } }

  calculatePriority {
    // Algorithmic code to calculate
    // priority of patients. } }

Actor Nurse {..}
Actor Clinician {..}
UserGroup Accountants {..}

Task PatientRegistration {
  Actor Nurse;
  Form Registration;
  KPI patientWaitTime;
  BusinessRules CostReimbursement;}

BusinessRule CostLimit {
  // Definition of Business Rule.. }

Scenario PatientAdmission {
  ContributesTo AdmitPatient;

  Triage ->
  Admit ->
  Discharge;}

Scenario PatientRegistration {..}

Scenario PatientDischarge { .. }

KPI PatientWaitTime {
  // Algorithmic code .. }

SoftGoal WaitTime {
  patientWaitTime;}

SoftGoal ReAdmission {..}

SoftGoal PatientSatisfaction {
  WaitTime & ReAdmission; }
```

Goals and SoftGoals

- Support for “AND” and “OR” decompositions.
- Goals are measured by KPIs



```
SoftGoal WaitTime {
  patientWaitTime;}

SoftGoal ReAdmission {..}

SoftGoal PatientSatisfaction {
  WaitTime & ReAdmission; }
```



```
Goal AdmitPatient{}

Class Patient {
  Integer age;
   Name;
   1 -- * Registration;

   patientStates {
     Admitted { .. }
     Re-Admitted { .. }
     Discharged { .. } } }

Form Registration {
  Patient.age mandatory;
  Patient.name mandatory;
   symptom optional;

  //state machine to define
  //behavior of the form.
  status {
     Open {
       submit [complete] -> Submitted
       close -> Closed; }

     Closed {
       entry/ {saveFormData();} }

     Submitted {
       reOpen -> Open; } }

   calculatePriority {
     // Algorithmic code to calculate
     // priority of patients. } }

Actor Nurse {..}
Actor Clinician {..}
UserGroup Accountants {..}

Task PatientRegistration {
   Actor Nurse;
   Form Registration;
   KPI patientWaitTime;
   BusinessRules CostReimbursement;}

BusinessRule CostLimit {
   // Definition of Business Rule.. }

Scenario PatientAdmission {
   ContributesTo AdmitPatient;

   Triage ->
   Admit ->
   Discharge;}

Scenario PatientRegistration {..}

Scenario PatientDischarge { .. }

KPI PatientWaitTime {
  // Algorithmic code .. }

SoftGoal WaitTime {
  patientWaitTime;}

SoftGoal ReAdmission {..}

SoftGoal PatientSatisfaction {
  WaitTime & ReAdmission; }
```

ROMPL Key Benefits

- Requirements are integrated within executable artifacts (no longer a separate artifact)
- Reduce or eliminate the need for requirements links.
- Broaden participation to include Business Analysts.
- Other?

Challenges in Adopting ROMPL

- ROMPL is in its incubation phase and requires further refinements
- Using different abstractions may introduce some problems
- Evaluation
    - Nurse on-Boarding Process (healthcare domain)

धन्यवाद
Hindi

多謝
TraditionalChinese

ขอบคุณ
Thai

Спасибо
Russian

Gracias
Spanish

Thank You
English

תודה
Hebrew

شكراً
Arabic

Merci
French

Obrigado
BrazilianPortuguese

Grazie
Italian

多谢
SimplifiedChinese

Danke
German

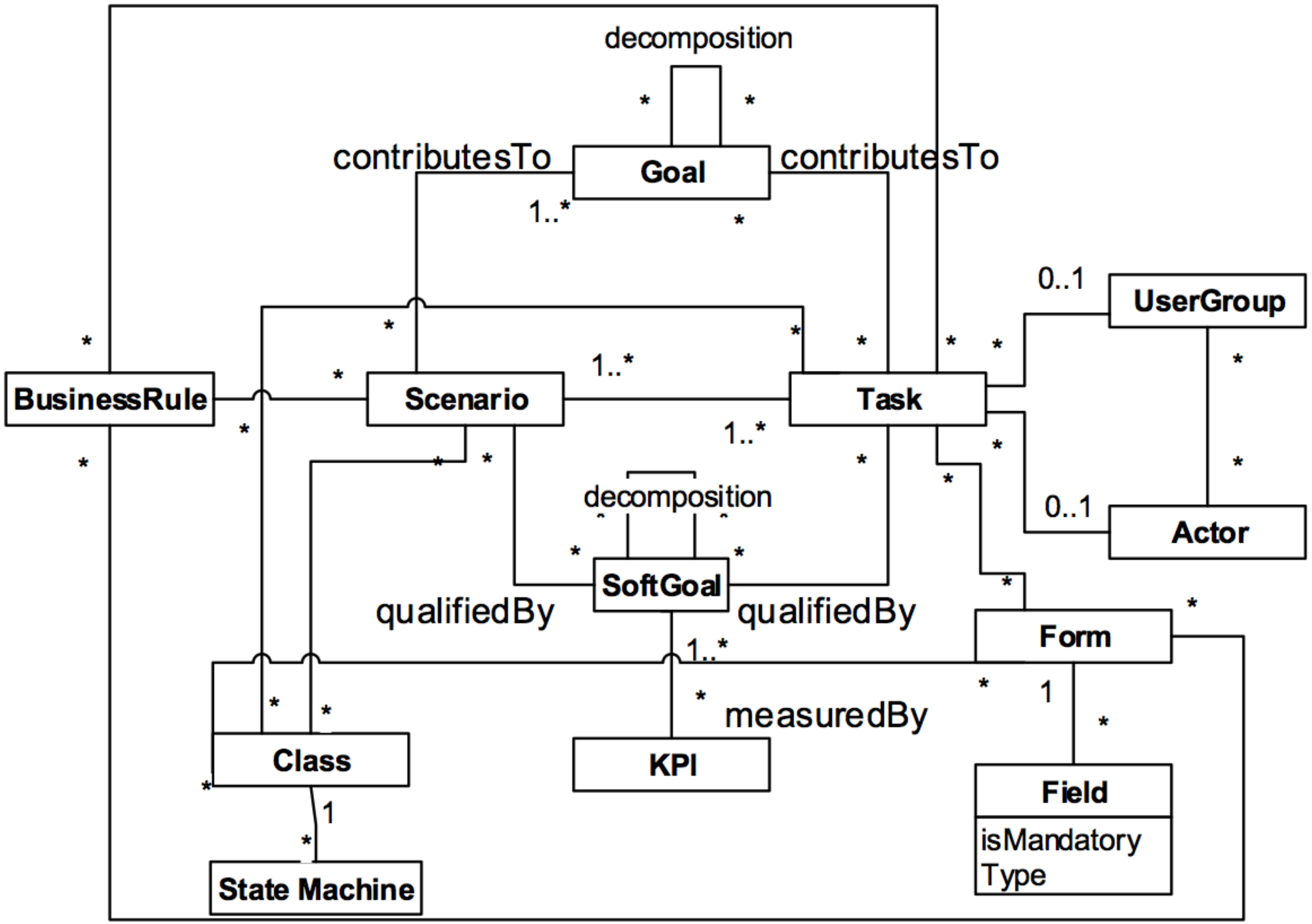
நன்றி
Tamil

ありがとうございました
Japanese

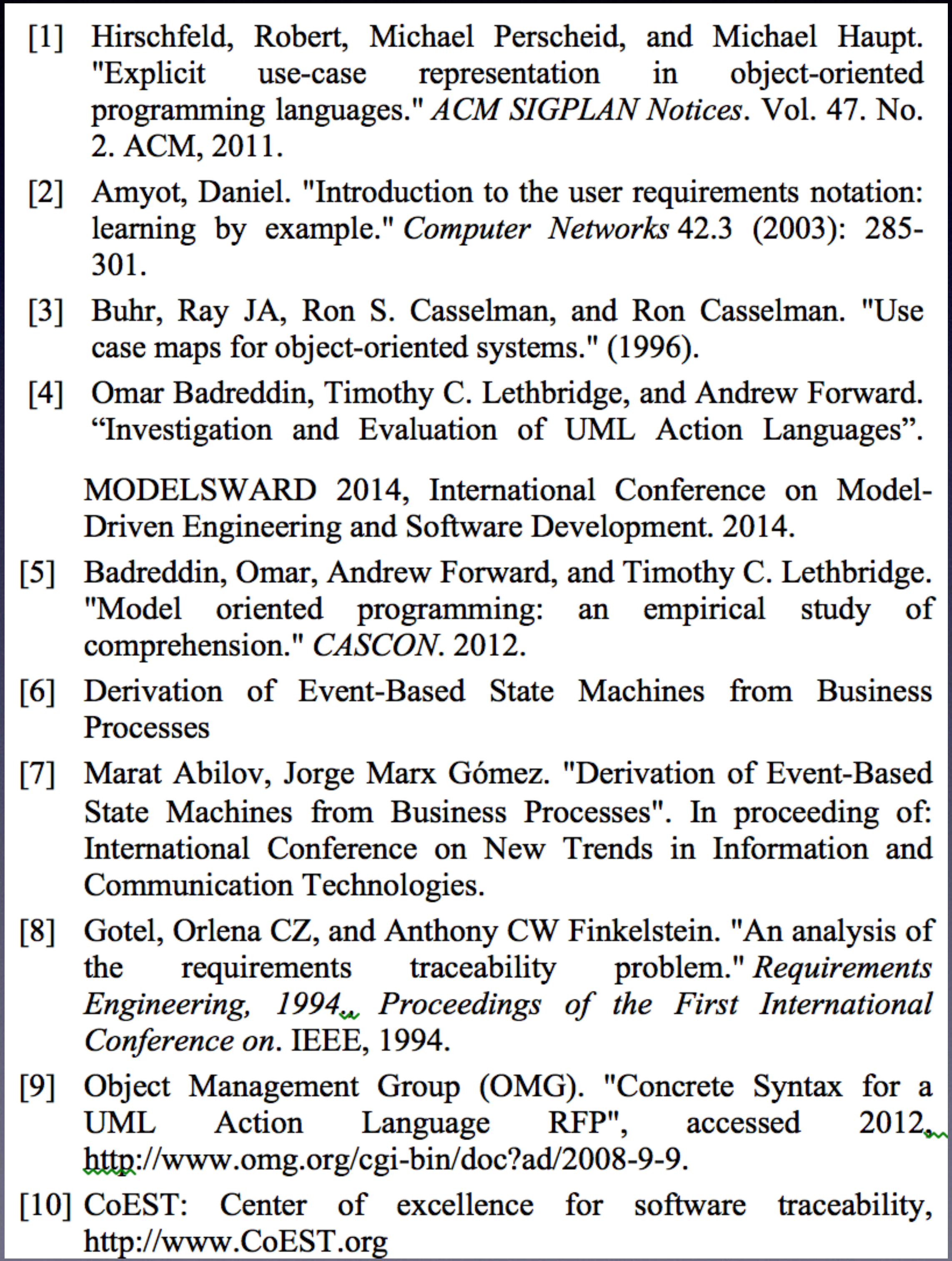
감사합니다

Literature Overview

- Connecting requirements to code has been attempted [1, 15].
- Attempts to link MDA and Requirements [18, 19]
- Probabilistic modeling of requirement traces [14]
- Computational reflection: the software system’s ability to dynamically observe and possibly modify its behavior [17]


decomposition
contributesTo
Goal
contributesTo
1..*
UserGroup
0..1
BusinessRule
Scenario
1..*
Task
1..*
decomposition
0..1
Actor
SoftGoal
qualifiedBy
qualifiedBy
Form
1..*
measuredBy
Class
KPI
Field
isMandatory
Type
1
1
State Machine

References



[1] Hirschfeld, Robert, Michael Perscheid, and Michael Haupt. "Explicit use-case representation in object-oriented programming languages." *ACM SIGPLAN Notices*. Vol. 47. No. 2. ACM, 2011.

[2] Amyot, Daniel. "Introduction to the user requirements notation: learning by example." *Computer Networks* 42.3 (2003): 285-301.

[3] Buhr, Ray JA, Ron S. Casselman, and Ron Casselman. "Use case maps for object-oriented systems." (1996).

[4] Omar Badreddin, Timothy C. Lethbridge, and Andrew Forward. "Investigation and Evaluation of UML Action Languages".

MODELSWARD 2014, International Conference on Model-Driven Engineering and Software Development. 2014.

[5] Badreddin, Omar, Andrew Forward, and Timothy C. Lethbridge. "Model oriented programming: an empirical study of comprehension." *CASCON*. 2012.

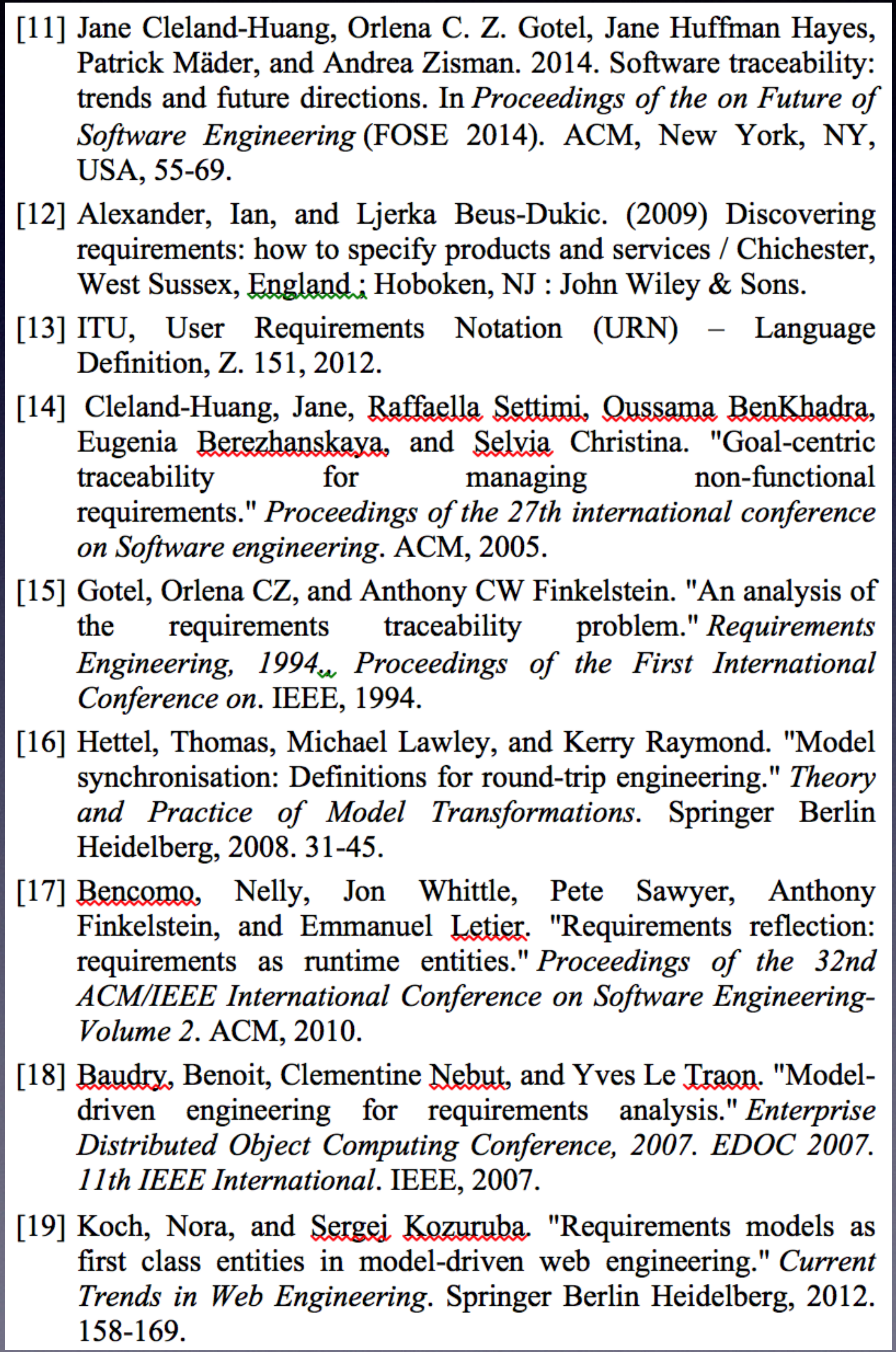
[6] Derivation of Event-Based State Machines from Business Processes

[7] Marat Abilov, Jorge Marx Gómez. "Derivation of Event-Based State Machines from Business Processes". In proceeding of: International Conference on New Trends in Information and Communication Technologies.

[8] Gotel, Orlena CZ, and Anthony CW Finkelstein. "An analysis of the requirements traceability problem." *Requirements Engineering, 1994., Proceedings of the First International Conference on*. IEEE, 1994.

[9] Object Management Group (OMG). "Concrete Syntax for a UML Action Language RFP", accessed 2012, http://www.omg.org/cgi-bin/doc?ad/2008-9-9.

[10] CoEST: Center of excellence for software traceability, http://www.CoEST.org



[11] Jane Cleland-Huang, Orlena C. Z. Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. 2014. Software traceability: trends and future directions. In *Proceedings of the on Future of Software Engineering* (FOSE 2014). ACM, New York, NY, USA, 55-69.

[12] Alexander, Ian, and Ljerka Beus-Dukic. (2009) Discovering requirements: how to specify products and services / Chichester, West Sussex, England ; Hoboken, NJ : John Wiley & Sons.

[13] ITU, User Requirements Notation (URN) – Language Definition, Z. 151, 2012.

[14] Cleland-Huang, Jane, Raffaella Settimi, Oussama BenKhadra, Eugenia Berezhanskaya, and Selvia Christina. "Goal-centric traceability for managing non-functional requirements." *Proceedings of the 27th international conference on Software engineering*. ACM, 2005.

[15] Gotel, Orlena CZ, and Anthony CW Finkelstein. "An analysis of the requirements traceability problem." *Requirements Engineering, 1994., Proceedings of the First International Conference on*. IEEE, 1994.

[16] Hettel, Thomas, Michael Lawley, and Kerry Raymond. "Model synchronisation: Definitions for round-trip engineering." *Theory and Practice of Model Transformations*. Springer Berlin Heidelberg, 2008. 31-45.

[17] Bencomo, Nelly, Jon Whittle, Pete Sawyer, Anthony Finkelstein, and Emmanuel Letier. "Requirements reflection: requirements as runtime entities." *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*. ACM, 2010.

[18] Baudry, Benoit, Clementine Nebut, and Yves Le Traon. "Model-driven engineering for requirements analysis." *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International*. IEEE, 2007.

[19] Koch, Nora, and Sergej Kozuruba. "Requirements models as first class entities in model-driven web engineering." *Current Trends in Web Engineering*. Springer Berlin Heidelberg, 2012. 158-169.