Scrum Conceptualization Using K-CRIO Ontology

Yishuai Lin, Vincent Hilaire, Nicolas Gaud, Abderrafiaa Koukam

Scrum Conceptualization Using K-CRIO Ontology

Yishuai Lin, Vincent Hilaire, Nicolas Gaud, and Abderrafiaa Koukam

Laboratoire Systèmes et Transports (SeT)
Université de Technologie de Belfort-Montbéliard (UTBM)
90010 Belfort cedex, France
vincent.hilaire@utbm.fr
+33 384 583 009
www.multiagent.fr

Summary. Many enterprises are on the path towards the automation of their business processes. To be fully efficient this kind of automation needs to rely on rich models of business processes. This paper presents an ontology, named K-CRIO, that allows the description of a specific kind of business processes: those that are dedicated to the design of a product. This ontology draws from organizational theories. The contribution of this paper is twofold. On the one hand, it described the key concepts and relationships of the k-CRIO ontology and on the other hand it illustrates this ontology by taking the example of the Scrum development process.

1.1 Introduction

Many enterprises are on the path towards the automation of their business processes. To be fully efficient this kind of automation needs to rely on rich models of business processes. This paper presents an ontological approach that allows the description of a specific kind of business processes: those that are dedicated to the design of a product. This ontology draws from organizational theories.

As stated in [3], an ontology of organizations ”is the first, fundamental and ineliminable pillar on which to build a precise and rigorous enterprise modeling”. Obviously, the field of ontologies for organizations is very broad, among the works in this domain one can cite [3, 8]. As the term organization covers a wide range of entities we have chosen to reduce our field of investigation. In our case the targeted organizations are those composed of human actors involved in the design of a product. These human actors follows a previously defined design process to frame their design activities. Even if this statement reduce our filed of investigation it is general enough to cover the description of many enterprises.

This ontology is the first building block of a software environment, based on muti-agent systems, that will support enterprises engaged in products design during their day-to-day works. Example of such support can be file sharing, enhanced communications, wikis, etc. The goal of this ontology is used to support knowledge management within the described organizations. More specifically, the ontology will provide means for annotating resources, reasoning, monitoring design processes, enabling searches and to proactively propose tips and proper contents. These resources and contents are those handled by the multi-agent system.

The presented approach is based upon the use of an existing organizational metamodel, namely CRIO [5], already used for the description of Multi-Agent Systems (MAS) organizations. In our case, the concepts of this metamodel are used to model human activities. The concepts and relationships of this metamodel are represented by the K-CRIO ontology.

The chosen ontology language is OWL [12] as it is widely used and allows the representation of the underlying concepts such as human interactions by using the OWL-WS sub-ontology [2].

This paper is organized as follows: Section 1.2 introduces the background of our work. Section 1.3 details the application of K-CRIO to the SCRUM methodology. Section 1.4 reviews and discusses related works. Eventually, Section 1.5 concludes and describes some future work directions.

1.2 Background

1.2.1 K-CRIO

In the following, we will try to single out which are the main concepts of an ontology that represent organizations. The K-CRIO ontology is presented in Figure 1.4.

The targeted organizations are those dedicated to product design. We have then defined a concept named DesignObject (*owl:class*) which is the root of all possible products that an organization can produce.

An organization can be seen as a set of interacting entities: sub-organizations or roles, which are regulated by social rules and norms.

- With respect to the ontology, an organization may be seen as a concept connected to other concepts by various kinds relationships, such as *hierarchical* relations between organizations and sub-organizations, or *composed of* relation between an organization and its roles.
- With respect to the human processes in enterprises, an organization may be considered as a collective global system able to achieve particular goals through its collaborative members.

Using OWL, the concept of organization may be specified as following: Organization is an *owl:class* which may be linked to sub-organizations with "isSubOrganizationOf" (*owl:ObjectProperty*)and "includes" (*owl:ObjectProperty*)

a collection of Roles (an *owl:class*), with "provided" (*owl:ObjectProperty*) Capacity (as well as an *owl:class*) and "hascontext" (*owl:ObjectProperty*) Ontology (an *owl:class*). The latter class defines a specific context for the concerned organization. Additionally, organizations are linked by "isThePlaceOf" (*owl:ObjectProperty*) to Interaction (*owl:class*) which conceptualize interactions occuring in the organization context. It may be expressed as detailed in Figure 1.1.
In the rest of this paper, we denote these definition with logic language:

Organization(X) means that X is an Organization.

DesignObject(X) means that X is a DesignObject.

Role(X) means that X is a Role.

Capacity(X) means that X is a Capacity.

Ontology(X) means that X is an Ontology.

Interaction (X) means that X is an Interaction.

isSubOrganizationOf (X,Y) means that X is a sub-Organization of Y and {*X: Organization(X), Y:Organization(Y)*}.

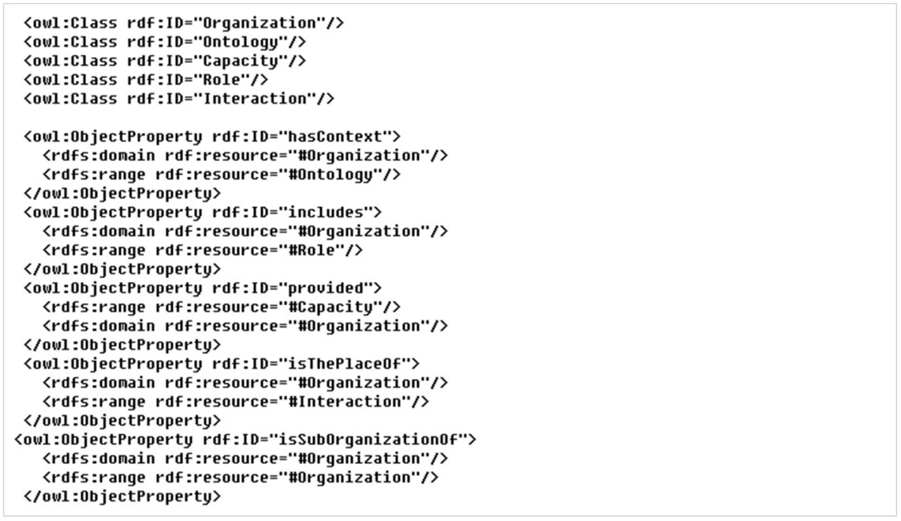
includes (X,Y) means that X includes Y and {*X: Organization(X), Y: Role(Y)*}.

provided (X,Y) means that X includes Y and {*X: Organization(X), Y: Capacity(Y)*}.

hasContext (X,Y) means that X hasContext Y and {*X: Organization(X), Y: Ontology(Y)*}.

isThePlaceOf (X,Y) means that X is the place of Y happening and {*X: Organization(X), Y: Interaction(Y)*}.

We also define some necessary restrictions between Organization and associated classes, which are represented by qualified cardinality restrictions in OWL (cardinality constraints including *owl:cardinality, owl:minCardinality, owl:maxCardinality* [16]) and axiomatized by logic language.
Specially, one Organization may have an unspecified number of sub-organizations (zero or more). Moreover, the *ObjectProperty: isSubOrganizationOf* is transitive objectProperty. For example, if we consider a university as one Organization, different departments may be seen as its sub-organizations. In the other side, one department is usually composed of diverse majors, which may be seen



```
 <owl:Class rdf:ID="Organization"/>
 <owl:Class rdf:ID="Ontology"/>
 <owl:Class rdf:ID="Capacity"/>
 <owl:Class rdf:ID="Role"/>
 <owl:Class rdf:ID="Interaction"/>

 <owl:ObjectProperty rdf:ID="hasContext">
   <rdfs:domain rdf:resource="#Organization"/>
   <rdfs:range rdf:resource="#Ontology"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="includes">
   <rdfs:domain rdf:resource="#Organization"/>
   <rdfs:range rdf:resource="#Role"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="provided">
   <rdfs:range rdf:resource="#Capacity"/>
   <rdfs:domain rdf:resource="#Organization"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:ID="isThePlaceOf">
   <rdfs:domain rdf:resource="#Organization"/>
   <rdfs:range rdf:resource="#Interaction"/>
 </owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isSubOrganizationOf">
   <rdfs:domain rdf:resource="#Organization"/>
   <rdfs:range rdf:resource="#Organization"/>
 </owl:ObjectProperty>
```

Fig. 1.1. Organization in K-CRIO

as sub-organizations of this department. Because of transitivity, these majors are also the sub-Organizations of the university. Expressed by owl:Restriction, the sub-Organization "isSubOrganizationOf" its Organization with minCardinality 0 (*owl:restriction*). The logic expression of transitivity of *ObjectProperty: isSubOrganizationOf* as:

isSubOrganizationOf (O_1, O_2), *isSubOrganizationOf* $(O_2, O_3) \rightarrow$ *isSubOrganizationOf* (O_1, O_3)

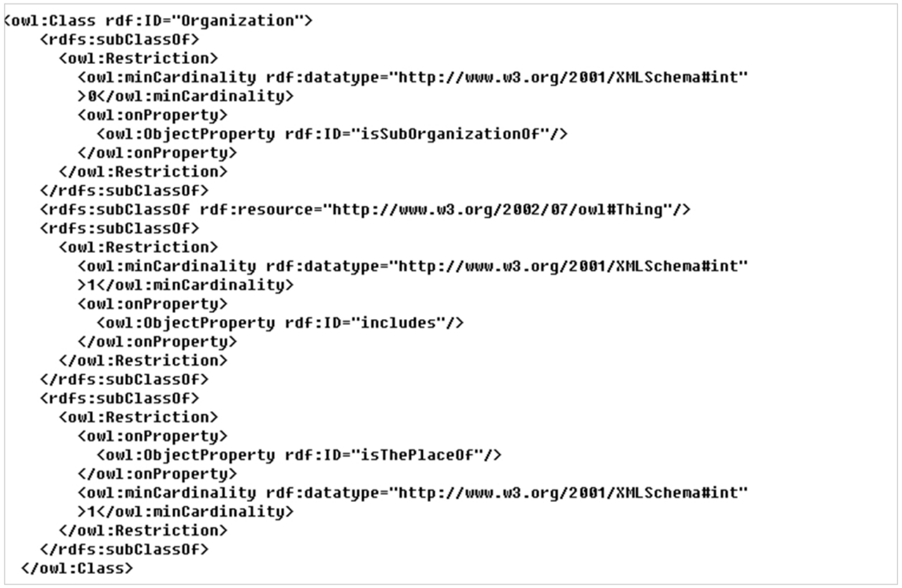
Moreover, one Organization must include one Role at least (inversely, one Role must be included by one Organization at least) and in which one or more interactions must be occurred. That means, describing with owl:Restriction, Organization "includes" Role with minCardinality 1 (*owl:restriction*) and "isThePlaceOf" interaction happening with minCardinality 1 (*owl:restriction*). The logic axiomatization expression as:

$$\forall\ O\ Organization\ (O) \rightarrow \exists\ R\ \{\ Role\ (R) \bigwedge includes\ (O,R)\ \}$$

$$\forall\ R\ Role(R) \rightarrow \exists\ O\ \{\ Organization\ (O) \bigwedge includes\ (O,R)\ \}$$

$$\forall\ O\ Organization\ (O) \rightarrow \exists\ I\ \{\ Interaction\ (I) \bigwedge isThePlaceOf\ (O,I)\ \}$$

It is detailed in the Figure 1.2 that these property restrictions used the xsd namespace declaration made in the header element to refer to the XML Schema document.



```
<owl:Class rdf:ID="Organization">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >0</owl:minCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="isSubOrganizationOf"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="includes"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="isThePlaceOf"/>
      </owl:onProperty>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Fig. 1.2. Restrictions of Organization in K-CRIO

A role may identify a person, an activity or a service which is a necessary part to achieve social objectives (goals of its organization). In order to fulfill this common target, each role may have specific individual capacities. An Organization (an *owl:class*) "includes" (*owl:ObjectProperty*) Roles (an *owl:class*) which may "required" (*owl:ObjectProperty*) Capacity (an *owl:class*):

required (X,Y) means X required Y and $\{X\text{: }Role(X),\ Y\text{: }Capacity(Y)\}$.

Moreover, Interaction (*owl:class*) occurring in Organization "hasParticipants" (*owl:ObjectProperty*) that are Roles.

hasParticipants (X,Y) means the at X hasParticipants Y happening and $\{X\text{: }Interaction(X),\ Y\text{: }Role(Y)\}$.

A capacity is a know-how and ability, which may be considered as an interface between the role and the role-players. Capacities are required by the role and provided by the organization to define the behaviors of other roles. With the respect of K-CRIO, beside the relationship with Organization and Role described above, Capacity is represented by an *owl:class*. This class is related to some *ContextOntologyElement* (which are parts of the Organization Ontology defining it context) divided in two sets "input" and "output" (both are *owl:ObjectProperty*). The Capacity class is also related to properties which are conceptualized by the Predicate concept. A predicate is an assertion on a property of some *ContextOntologyElement*. There are two types of relationships between Capacity and Predicate. The first type is named "*requires*" and represents the properties that are required by the capacity. The second type is named "*ensures*" and represents the effects of the capacity.

Interactions are mandatory for modeling human processes. The aim of an interaction in this context is to achieve a goal or to contribute to a goal of one organization. Interactions may be divided into *Casual Interaction* and *Formalized Interaction*. The former ones depict interactions between roles which are not defined initially in the model and that can take place in a non determined fashion. Inversely, formalized interactions identify how different roles belonging to the same organization can interact with each other so as to achieve the common goal of the organization and in the meanwhile produce *DesignObjects* of any sort. Interactions may then be seen as a description of possible patterns of actions and exchanges between the participant roles that is very similar to a workflow. In order to conceptualize *FormalizedInteraction*, we have chosen to reuse and improve an existing ontology, OWL-WS, dedicated to the description of workflows [2] and inspired from OWL-S [11]. OWL-WS is based on the assumption that a workflow is a kind of complex service and therefore it can be represented in OWL-WS as a full OWL-S Service. This service can be a simple one or composed of simpler services using the OWL-S control constructs such as: *Sequence*, *RepeatUntil*, *Split*, etc.

A process may be an atomic process which is a description of one (possibly complex) message and returning one (possibly complex) message in response. A process may also be a composite which means that it can be divided into atomic processes describing a behavior (or a set of behaviors) sending and receiving a series of messages. If we consider the formalized interaction as a composite process owning an overall effect, the the entire process must be performed in order to achieve that objective. Moreover, the definition of formalized interaction express the different ways of executing this interaction and correlative results. Let's take the example of a selling service, represented by one organization named selling service including two roles: Client and Bank Service. The formalized interaction details how to perform this trade between these two roles (exchanging messages).

- The initial state is when the Client inputs the Credit Card Number and code,which is a message sent to Bank Service.
- When Bank Service receives a message from the Client role, there should be a control accompanying different situations as
  - Both datum are right, Bank Service returns confirm message to Client;
  - One datum is invalid, Bank Service returns error message to Client, simultaneously, the state of Client returns to original state (waiting for an input of the Credit Card Number and Password and waiting new answer of Bank Service);

- This sequence is repeated until the Client receives the confirm message from Bank Service, he can then send Verify Paying message to Bank Service;
- Bank Service validate the charge following the message Verify Paying from Client.

From the point of view of K-CRIO, *CasualInteraction* and *FormalizedInteraction* are represented by two *owl:class* both subclasses of *Interaction* (an *owl:class*), which are related by "hasParticipants" (*owl:ObjectProperty*) to Roles:

CasualInteraction (X) means X is CasualInteraction.

FormalizedInteraction (X) means X is FormalizedInteraction.

Interaction (X) $\equiv$ *CasualInteraction (X)* $\bigcup$ *FormalizedInteraction (X).*

The *FormalizedInteraction* class is related by the "*produces*" (*owl:ObjectProperty*) relationships to the "*DesignObject*" (an *owl:class*) concept:

produces (X,Y) means X produces Y and {*X: FormalizedInteraction(X), Y: DesignObject(Y)*}.

Moreover, concerning the state of each formalized interaction, we defined FormalizedInteraction is in (an ObjectProperty) some State (an owl:class), which has three sub-classes: NotStart, Doing, Done. In certain situations, a formalized interaction has its Pre-Interaction (hasPre-Interaction is an objectProperty, the domain and range of which are both FormalizedInteraction):

at (X,Y) means X is at Y state and {*X: FormalizedInteraction(X), Y: State(Y)*}.

hasPre-Interaction (X,Y) means X has pre-interaction Y and {*X: FormalizedInteraction(X), Y: FormalizedInteraction(Y)*}.

In order to test whether the interaction is executed following the schedule or not, we defined an owl:class: Time which is related with FormalizedInteraction by the objectProperty: follows. Specially, Time has four sub-classes: BeginningTime, EndTime, RealBeginningTime, RealEndTime, in which, BeginningTime and EndTime expresses the planning schedule and RealBeginningTime and RealEndTime expresses the real schedule. Comparing the subtraction of two types of beginning time and end time, we could express whether the process is earlier or later than the schedule actually:
For *Follows(FormalizedInteraction, BeginningTime)* $\bigwedge$ *(FormalizedInteraction, EndTime)*$\bigwedge$*(FormalizedInteraction, RealBeginningTime)*$\bigwedge$*(FormalizedInteraction,*

RealEndTime):

(EndTime - BeginningTime) > *(RealEndTime - RealBeginningTime)* means the FormalizedInteraction is earlier than the schedule.

(RealEndTime - RealBeginningTime) > *(EndTime - BeginningTime)* means the FormalizedInteraction is later than the schedule.

In summary, the whole K-CRIO Ontology is presented in Figures 1.3 and 1.4.

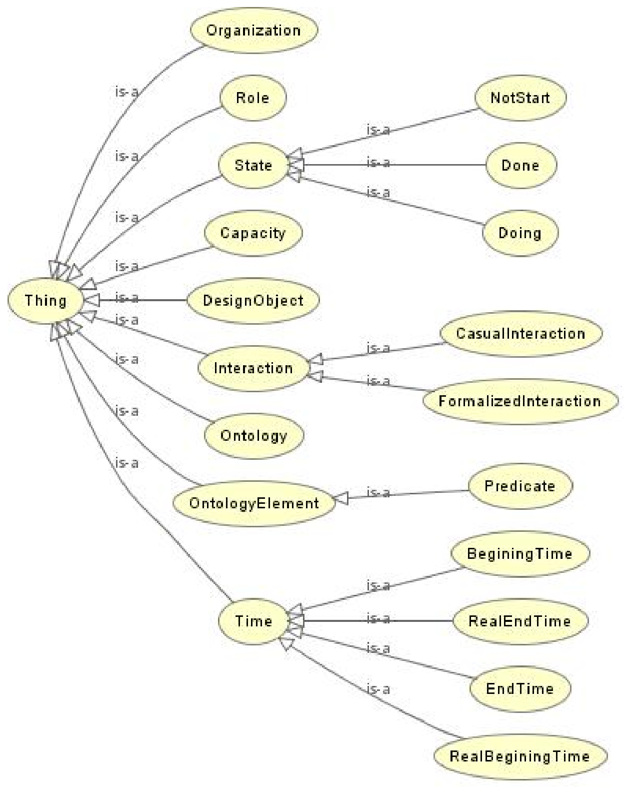


Fig. 1.3. K-CRIO taxonomy

1.2.2 Introduction to Scrum

Scrum is an iterative, incremental framework for projects and products or application developments and is often seen as an agile software development

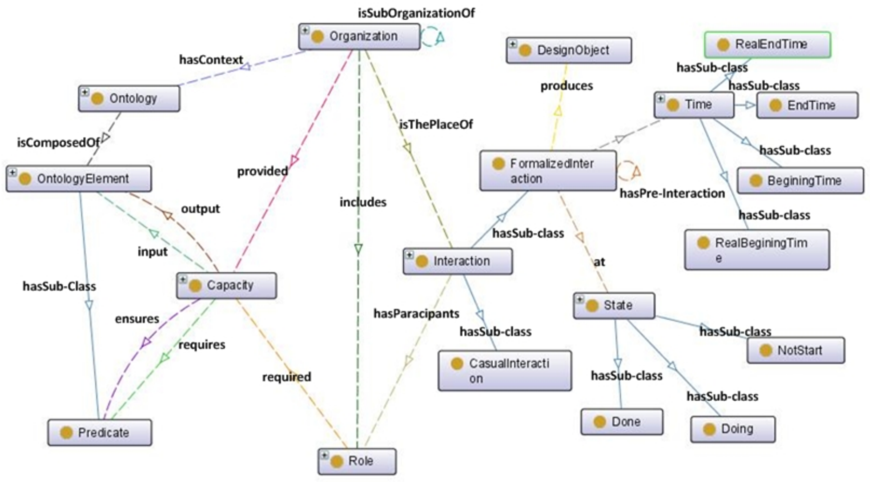


Fig. 1.4. K-CRIO Ontology

process. Scrum structures development in cycle of work called Sprints (or iterations). These iterations last no more than one month each, and take place one after the other without pause. The people involved in Scrum could be divided into two groups. The first group denotes the Scrum team and has a figurative name : ”Pig”, where are three core roles: The Product Owner, The Scrum Master and The Team. All these roles are committed to the project in the Scrum process, who are the ones producing the product (objective of the project). The other group delegates Ancillary people, relatively, which is named ”Chicken”. Precisely, the ancillary people in Scrum are those with no formal role and infrequent involvements in the Scrum process and must nonetheless be taken into account, such as some Stakeholders, Customers, Vendors or Managers (that represents a kind of people who will set up the environment for product development).

The Scrum process could be presented by the Figure 1.5 from [7]. In this figure, we could see that during the first step, the Product Owner needs to communicate with all the Stakeholders in order to transform each User Story into one item or one feature in the Product Backlog, which is the basic document containing prioritized descriptions which will be referred by the Scrum Team during the Scrum Process. In a second step, before the beginning of each Sprint, a cross-functional team selects items (customer requirements) from the prioritized list (Product Backlog) [7]. This phase occurs in the Sprint Planning Meeting. The team should commit to complete these items by the end of the Sprint, which may be called task and represented in the Sprint Backlog. More informations about Scrum could be found in [7, 17].

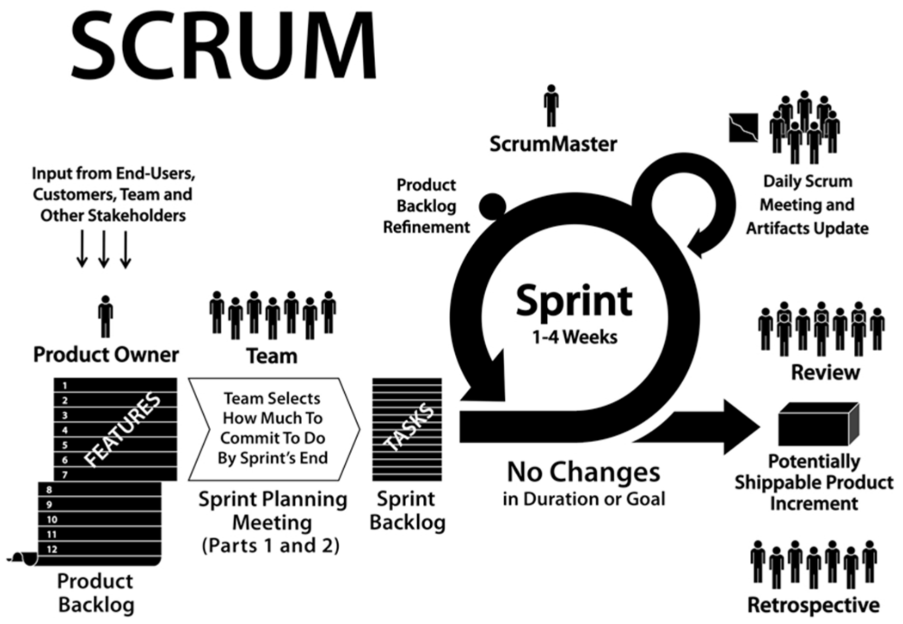

SCRUM
Input from End-Users, Customers, Team and Other Stakeholders
Product Owner
Team
ScrumMaster
Product Backlog Refinement
Daily Scrum Meeting and Artifacts Update
Sprint
1-4 Weeks
Review
FEATURES
TASKS
Team Selects How Much To Commit To Do By Sprint's End
Sprint Planning Meeting (Parts 1 and 2)
Sprint Backlog
No Changes in Duration or Goal
Potentially Shippable Product Increment
Product Backlog
Retrospective

Fig. 1.5. The process of Scrum

1.3 Scrum represented by K-CRIO

In this section, we will use the K-CRIO Ontology [10] to describe human roles and interactions during the Scrum process, containing actors, their capacities and how these roles interacts with others in each phase during a Scrum process. Based on general understand of Scrum above, we could see that the whole Scrum contains diverse kinds of people working together with a specific interactive mode in order to achieve a common goal, such as delivering a product or exploring a project. Following the definition of K-CRIO, Scrum may be seen as one Organization in K-CRIO, in which there are smaller groups also seen as Organizations (sub-organization of Scrum), like the groups "Pig" and "Chicken", or some explicit people seen as Role, like Scrum Master, Product Owner,etc. The expression of definition with logic language as:

***Scrum**: Organization (Scrum)* $\bigwedge$ *isSubOrganization (Pig, Scrum)* $\bigwedge$ *isSubOrganization (Chicken, Scrum)*

- In the *Organization: Scrum*, the group "Pig" (Scrum Team) aims at releasing the product following the requirement of customers and vendors through its members contributions. Therefore, the group "Pig" (Scrum

Team) may be seen as the *Organization: "Pig"*(*Organization: Scrum Team*) in K-CRIO, which is sub-Organization of *Organization: Scrum.* As three significant members of "Pig"(Scrum Team), the Product Owner (PO), the Scrum Master (SM) and the Developing Team (DT) fulfill the objective of *Organization: "Pig"* through accomplishing their jobs. Therefore, we could define "Pig" as:

Pig(ScrumTeam)*: Organization (Pig)* $\bigwedge$ *isSubOrganization (Pig, Scrum)* $\bigwedge$ *isSubOrganization (DT, Pig)* $\bigwedge$ *includes (Pig, PO)* $\bigwedge$ *includes (Pig, SM)* $\bigwedge$ provided (Pig, Capacities for PO) $\bigwedge$ provided (Pig, Capacities for SM)

Because of the transitivity of isSubOrganization, there is:

isSubOrganization (Pig, Scrum), isSubOrganization (DT, Pig) $\rightarrow$ *isSubOrganization (DT, Scrum)*

Considering about the two Roles of the Pig Group, Product Owner represents the voice of the customer and writes customer-centric items (typically user stories), prioritizes them, and adds them to the product backlog, which is accountable for ensuring that the Team delivers value to the business:

PO*: Role (PO)* $\bigwedge$ *includes (Pig, PO)* $\bigwedge$ *provided (Pig, Capacities for PO)*

As an enforcer of rules, Scrum Master is accountable for removing impediments to the ability of the team to deliver the sprint goal/deliverables and ensures that the Scrum process is used as intended:

SM*: Role (SM)* $\bigwedge$ *includes (Pig, SM)* $\bigwedge$ *provided (Pig, Capacities for SM)*

Differently, as one of necessary members in the group "Pig", Developing Team is typically made up of 5 to 9 developers with cross-functional skills who do the actual work (analyze, design, develop, test, technical communication, document, etc.) for releasing the product. For this reason, we may see Product Owner and Scrum Master as two Roles included in the *Organization: "Pig"* and the Developing Team is one Organization which is sub-Organization of *Organization: "Pig"*. Additionally, the Developer in Developing Team may be seen as Role in K-CRIO, which is included in *Organization: Developing Team.*(Sometimes, Product Owner and Scrum Master may be as members in the Developing Team [1, 7], that means *Organization: Developing Team* may include *Role: Product Owner* and *Role: Scrum Master* in certain situation. However, generally these two Roles are out of Developing Team.):

***DT**: Organization (DT) $\bigwedge$ isSubOrganization (DT, Pig) $\bigwedge$ includes (DT, Developer) $\bigwedge$ provided (DT, Capacities for Developer)*

***Developer**: Role (Developer) $\bigwedge$ includes (DT, Developer) $\bigwedge$ required (Developer, Capacities for Developer)*

Following the analysis above, *Organization: "Pig"* and *Organization: Developing Team* provide the following capacities required by the roles composing it.

- *Organization: "Pig"* provides:
  - *Capacity: Representing interests of stakeholder*, *Capacity: Writing User Stories*, *Capacity: Prioritizing Users Stories*, *Capacity: Maintaining the Product Backlog*, etc. (required by *Role: Product Owner*);
  - *Capacity: Protecting and keeping team focused on its tasks*, *Capacity: Enforcing rules,Removing impediments*, *Capacity: Ensuring Scrum Process used as intended*,etc. (required by *Role: Scrum Master*);
- *Organization: Developing Team* provides: *Capacity: Cross-functional developing skills*, *Capacity: Delivering the Product* (required by *Role: Developer*).

- Similarly, in the *Organization: Scrum*, the group "Chicken" (Ancillary People) is a group of people including Stakeholder and Manager. This group may be seen as the *Organization: "Chicken"* (*Organization: Ancillary People*) in K-CRIO, which is sub-Organization of *Organization: Scrum.* Especially, the Stakeholder represents people for whom the project will produce the agreed-upon benefits, which justify its production, such as the Customer and the Vendor. Furthermore, Manager is the people who will set up the environment for product development. Consequently, *Organization: "Chicken"* includes one *Role:Manager* and has one sub-Organization *Organization: Stakeholder*, which includes *Role: Customer* and *Role: Vendor*:

  ***Chicken**: Organization (Chicken) $\bigwedge$ includes (Chicken, Manager) $\bigwedge$ isSubOrganization (Stakeholder, Chicken) $\bigwedge$ provided (Chicken, Capacities for Manager)*

  ***Stakeholder**: Organization (Stakeholder) $\bigwedge$ isSubOrganizationOf (Stakeholder, Chicken) includes (Stakeholder, Customer) $\bigwedge$ includes (Stakeholder, Vendor) $\bigwedge$ provided (Stakeholder, Capacities for Customer) $\bigwedge$ provided (Stakeholder, Capacities for Vendor)*

  Moreover, we could reason:

isSubOrganization (Chicken, Scrum), isSubOrganization (Stakeholder, Chicken) $\rightarrow$ *isSubOrganization (Stakeholder, Scrum)*

Finally, we denote various capacities supplied to corresponding roles.

– *Organization: "Chicken"* provides *Capacity: Setting up environment for product development*, which is required by *Role: Manager.*
– *Organization: "Stakeholder"* provides *Capacity: Enabling project* required by both *Role: Customer* and *Role: Vendor.*

Following K-CRIO definitions, the above description may be expressed as presented in Figure 1.6. Organization, Role and Capacity are represented in purple, black and green respectively. Additionally, as an example describing how to define these elements by OWL, Figure 1.7 expresses *Role: Scrum Master* by OWL.

So far we have declared relevant Organizations, Roles and Capacities in Scrum. In order to describe the whole Scrum process, we will state the interactions occurring during the different Scrum phases. Interactions in K-CRIO may be considered as Casual Interaction or Formalized Interaction separately. Chat is a kind of Casual Interaction between different persons (Roles), other examples may be Exchanging Mail or Joining Conference Meeting, etc. It is Formalized Interactions of Scrum that produces relative *DesignObjects.* In the process of Scrum, we maybe define the following objects as *DesignObject*: *Product, Product Backlog, a release of its sprint, Sprint Backlog, Updated Burndown Chart* and represent Formalized Interaction by an OWL-WS process.

As sketched by Figure 1.8, the whole Scrum process may be seen as a Composite Process in order to produce the *DesignObject: Product*, which is composed by the Control Construct *Sequence*: a Composite Process: *Articulate Product Vision* (also called *The Pregame Phase*), a Composite Process: *The Game Phase* and a Composite Process: *The Postgame Phase* [17]. In all the figures describing Processes, a single rectangle means an Atomic Process and a double rectangle means a Composite Process.

Precisely, *Articulate Product Vision* may be seen as a Composite Process with the Control Construct *Sequence* (showing in the Figure 1.9): an Atomic Process: *Get Requirement from Stakeholder*, in which the Product Owner communicates with the stakeholder of the product for collecting User Stories and an Atomic Process: *Make Product Backlog* in order to output the *DesignObject: Product Backlog.*

The Figure 1.10 details *The Game Phase*, which may be seen as a composite process with the Control Construct *Repeat-While* for producing the *DesignObject: Product.* The Condition *Has Next Sprint* controls the loop and is initialized to true. While *Has Next Sprint* is true, the processes: *Sprint Planning Meeting*, *Sprint*, *Sprint Review Meeting* and *Sprint Retrospective Meeting* are executed orderly by the Control Construct *Sequence* and return

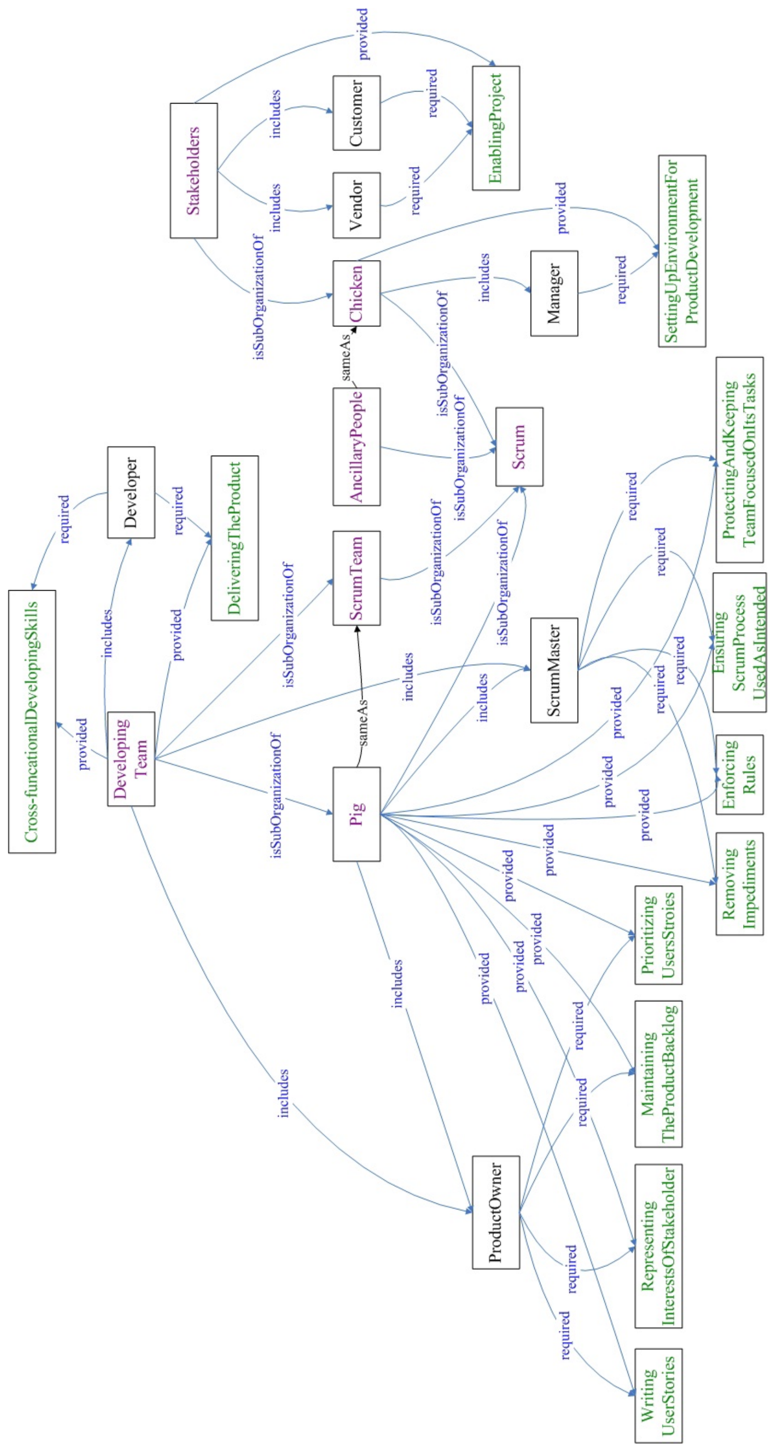
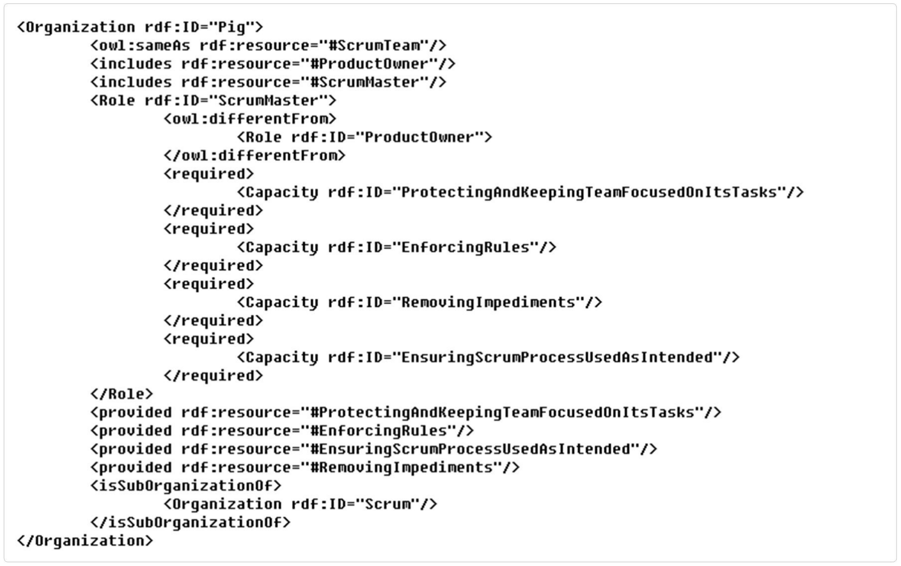


Fig. 1.6. Scrum with K-CRIO



```
<Organization rdf:ID="Pig">
        <owl:sameAs rdf:resource="#ScrumTeam"/>
        <includes rdf:resource="#ProductOwner"/>
        <includes rdf:resource="#ScrumMaster"/>
        <Role rdf:ID="ScrumMaster">
                <owl:differentFrom>
                        <Role rdf:ID="ProductOwner">
                </owl:differentFrom>
                <required>
                        <Capacity rdf:ID="ProtectingAndKeepingTeamFocusedOnItsTasks"/>
                </required>
                <required>
                        <Capacity rdf:ID="EnforcingRules"/>
                </required>
                <required>
                        <Capacity rdf:ID="RemovingImpediments"/>
                </required>
                <required>
                        <Capacity rdf:ID="EnsuringScrumProcessUsedAsIntended"/>
                </required>
        </Role>
        <provided rdf:resource="#ProtectingAndKeepingTeamFocusedOnItsTasks"/>
        <provided rdf:resource="#EnforcingRules"/>
        <provided rdf:resource="#EnsuringScrumProcessUsedAsIntended"/>
        <provided rdf:resource="#RemovingImpediments"/>
        <isSubOrganizationOf>
                <Organization rdf:ID="Scrum"/>
        </isSubOrganizationOf>
</Organization>
```

Fig. 1.7. *Role: Scrum Master* in OWL

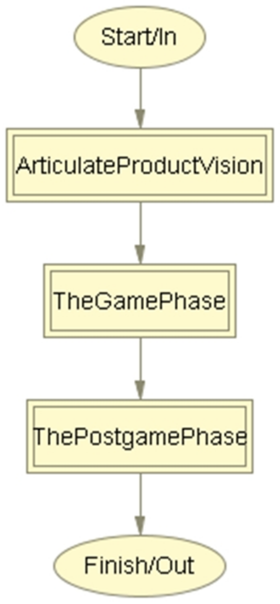


Fig. 1.8. A Composite Process: *Scrum Process*

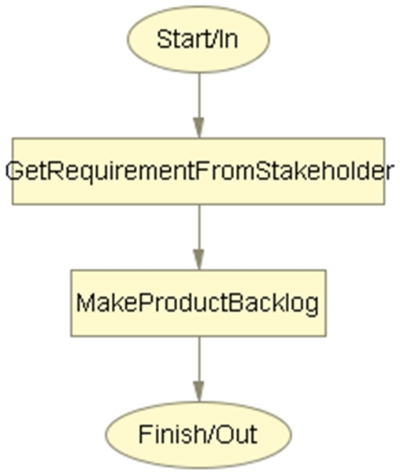


Fig. 1.9. A Composite Process: *Articulate Production Vision*

a new value of *Has Next Sprint*. If *Has Next Sprint* is false, the Composite Process: *The Game Phase* is finished.

Sprint Planning Meeting may be seen as a composite process aiming at producing the *DesignObject: Sprint Backlog* as the Figure 1.11, which is composed of two sequential Atomic Processes: *Sprint Planning Meeting_PartOne* which is for discussing what items of product are chosen for being realized in this Sprint, and *Sprint Planning Meeting_PartTwo* which is focusing on how to finish these tasks.

Sprint may be seen as a composite process with the Control Construct *Repeat-While* as the Figure 1.12. Every sprint is producing the *DesignObject: a release of its sprint*. The Condition *Sprint Is Not Finished* controls the loop and is initialized to true. While *Sprint Is Not Finished* is true, the Atomic Process: *Daily Scrum* (producing *DesignObject: Updated Burndown Chart*) is executed and return a new value of *Sprint Is Not Finished*. If *Sprint Is Not Finished* is false, the Composite Process *Sprint* is finished and *Sprint Review Meeting* and *Sprint Retrospective Meeting* are executed sequentially. Moreover, in the Figure 1.13, we use this simple formalized interaction, the Composite Process: *Sprint* to express defining the Formalized Interaction by OWL-WS.

Eventually, *The Postgame Phase* may be seen as a composite process, which may be composed of the processes: *Integration Testing*, *Writing User Document*, *Training Users*, *Marketing Material Preparation* and so on.

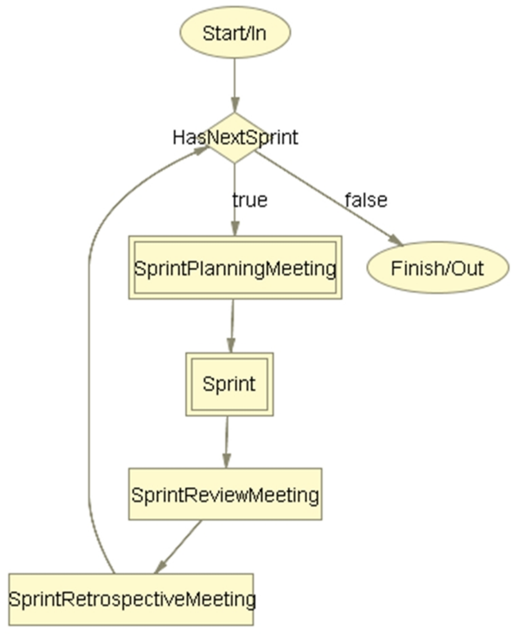


Fig. 1.10. A Composite Process: *The Game Phase*

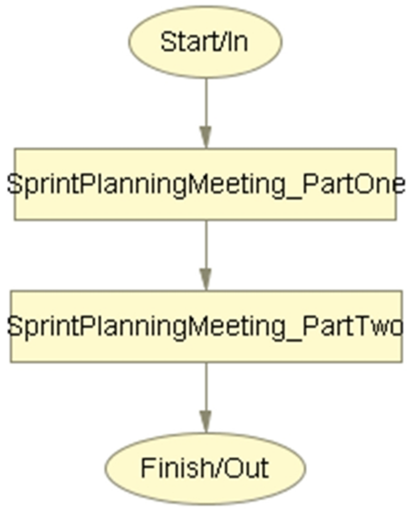


Fig. 1.11. A Composite Process: *Sprint Planning Meeting*

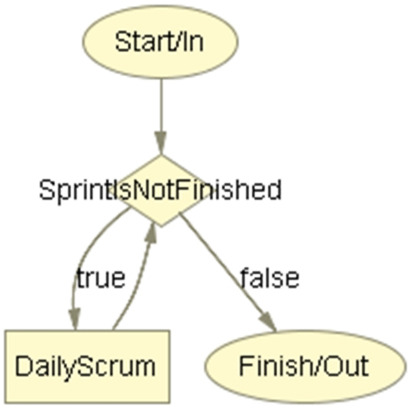
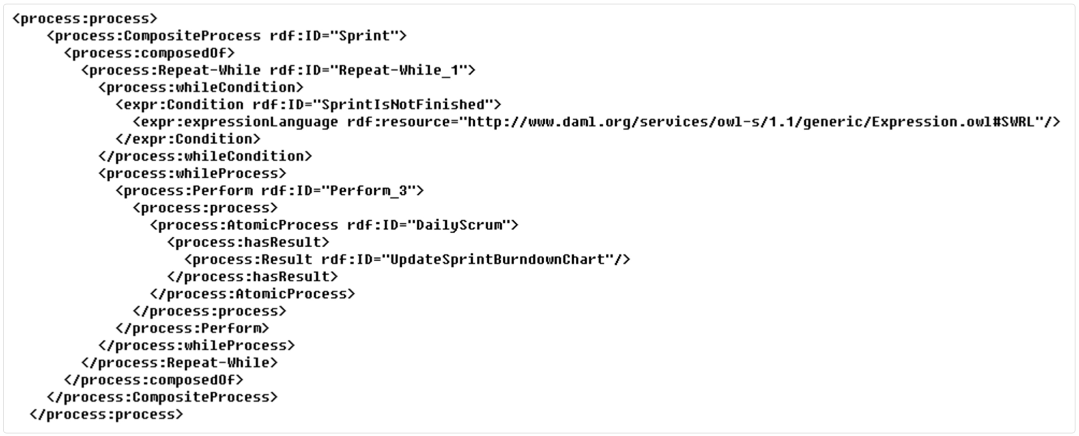


Fig. 1.12. A Composite Process: *Sprint*



```
<process:process>
    <process:CompositeProcess rdf:ID="Sprint">
      <process:composedOf>
        <process:Repeat-While rdf:ID="Repeat-While_1">
          <process:whileCondition>
            <expr:Condition rdf:ID="SprintIsNotFinished">
              <expr:expressionLanguage rdf:resource="http://www.daml.org/services/owl-s/1.1/generic/Expression.owl#SWRL"/>
            </expr:Condition>
          </process:whileCondition>
          <process:whileProcess>
            <process:Perform rdf:ID="Perform_3">
              <process:process>
                <process:AtomicProcess rdf:ID="DailyScrum">
                  <process:hasResult>
                    <process:Result rdf:ID="UpdateSprintBurndownChart"/>
                  </process:hasResult>
                </process:AtomicProcess>
              </process:process>
            </process:Perform>
          </process:whileProcess>
        </process:Repeat-While>
      </process:composedOf>
    </process:CompositeProcess>
  </process:process>
```

Fig. 1.13. A Composite Process: *Sprint* by OWL-WS

These concepts and relationships of Scrum represented by K-CRIO Ontology will be used in a tool for managing knowledge in really enterprise working under Scrum Method. The tool will be developed as Multi-Agent System in the environment Janus [6]. we now take some query examples to address how the ontology could help people managing knowledge during the activities.

- we want to know that one project is executed in which process at present? We suppose ***P*** is the process we want to find. Hence:
  P: FormalizedInteraction (P) $\bigwedge$ *at (P, Doing)* $\bigwedge$ *hasPre-Interaction* $(P, P_{\prime})$ $\bigwedge$ *at* $(P_{\prime}, Done)$

- we want to know which developers are late for the schedule during all the sprint process?

  We suppose ***D*** is the Developer we want to find. Hence:
  D: Developer (D) $\bigwedge$ *hasParticipants (Sprint, D)* $\bigwedge$ *follows (Sprint, BeginningTime)* $\bigwedge$ *follows (Sprint, RealBeginningTime)* $\bigwedge$ *follows (Sprint, EndTime)* $\bigwedge$ *follows (Sprint, RealEndTime)* $\bigwedge$ {*(RealEndTime - RealBeginningTime) > (EndTime - BeginningTime)*}

1.4 Related works

Formal structures such as OWL and RDFS have been used for Personal Information Management before; the PIMO model [15] aims to represent parts of the Mental Model necessary for tasks involving knowledge. The Mental Model is part of the cognitive system of a person. Subjective to a person, the mental model is individual and cannot be externalized thoroughly. The definition for a Personal Information Model (PIMO) is given as follow: A PIMO is a Personal Information Model of one person. It is a formal representation of parts of the users Mental Model. Each concept in the Mental Model can be represented using a Thing or a subclass of this class in RDF. Native Resources found in the Personal Knowledge Workspace can be categorized, and are occurrences of Thing. The vision is that a Personal Information Model reflects and captures a user's personal knowledge, e.g., about people and their roles, about organizations, processes, and so forth, by providing the vocabulary (concepts and their relationships) for expressing concepts as well as concrete instances. In other words, the domain of a PIMO is meant to be "all things and native resources that are pertinent for the user when doing work involving knowledge". Though "native" information models and structures are widely used, there is still much potential for a more effective and efficient exploitation of the underlying knowledge. Compared to the cognitive representations humans build, there are mainly two shortcomings in native structures:

- Model richness: current state of cognitive psychology assumes that humans build very rich models, representing not only detailed factual aspects, but also episodic and situational informations. Native structures are mostly taxonomy-oriented or keyword-oriented.
- Models coherence: though nowadays (business) life is very fragmented, humans tend to interpret situations as a coherent whole and have representations of concepts that are comprehensive across contexts. Native structures, on the other hand, often reflect the fragmentation of multiple contexts. They tend to be redundant (i.e., the same concepts at multiple places in multiple native structures). Frequently, inconsistencies are the consequence.

In brief, the PIMO shall mitigate the shortcomings of native structures by providing a comprehensive model on a sound formal basis.

Multilayered Semantic Social Network (MSSN) Model [4] proposes a multilayered semantic social network model that offers different views of common interests underlying a community of people, which is working within an ontology-based personalization framework [18], user preferences are represented as vectors $u_i = (u_{i,1}, u_{i,2}, ..., u_{i,N})$ where the weight $u_{i,j} \in [0, 1]$ measures the intensity of the interest of user i for concept c_j in the domain ontology, N being the total number of concepts in the ontology. Similarly, the objects d_k in the retrieval space are assumed to be described (annotated) by vectors $(d_{k,1}, d_{k,2}, ..., d_{k,N})$ of concept weights, in the same vector-space as user preferences. Based on this common logical representation, measures of user interest for content items can be computed by comparing preference and annotation vectors, and these measures can be used to prioritize, filter and rank contents (a collection, a catalog, a search result) in a personal way. The applicability of the proposed model to a collaborative filtering system is empirically studied. Starting from a number of ontology-based user profiles and taking into account their common preferences, the concept space domain is automatically clustered. With the obtained semantic clusters, similarities among individuals are identified at multiple semantic preference layers, and emergent, layered social networks are defined, suitable to be used in collaborative environments and content recommenders.

The AIC Model represents such a system as a tripartite graph with hyperedges [13]. The set of vertices is partitioned into the three (possibly empty) disjoint sets $A = \{a_1, a_2, ..., a_k\}$, $C = \{c_1, c_2, ..., c_i\}$, $I = \{i_1, i_2, ..., i_m\}$ corresponding to the set of actors (users), the set of concepts (tags, keywords) and the set of annotated objects (bookmarks, photos etc). It extends the traditional bipartite model of ontology (concepts and instances) by incorporating actors in the model. In a social tagging system, users tag objects with concepts, creating ternary associations between the user, the concept and the object. Thus the folksonomy is defined by a set of annotations $T \in A \times C \times I$. Such a network is most naturally represented as an hypergraph with ternary edges, where each edge represents the fact that a given actor is associated with a certain instance and a certain concept. In particular, the author defines the representing hypergraph of a folksonomy T as a (simple) tripartite hypergraph $H(T) = (V, E)$ where $V = A \cup C \cup I$, $E = \{\{a, c, i\} \mid (a, c, i) \in T\}$.

The MOISE+ Model [9] structure is built up in three levels: one is the behaviors that an agent playing a role is responsible for (individual), the other is the structure and interconnection of the roles with each other (social), and the last is the aggregation of roles in large structures (collective). In MOISE+, as in MOISE, three main concepts, roles, role relations, and groups, are be used to build, respectively, the individual, social, and collective structural levels of an organization. Furthermore, the MOISE original structural dimension is enriched with concepts such as inheritance, compatibility, cardinality, and sub-groups.

- **Individual level** is formed by the roles of the organization. A role means a set of constraints that an agent ought to follow when it accepts to enter a group playing that role. Following, these constraints are defined in two ways: in relation to other roles (in the collective structural level) and in a deontic relation to global plans (in the functional dimension). In order to simplify the specification, like in Object-Oriented (OO) terms, there is an inheritance relation among roles. If a role p' inherits a role p (denoted by $p \subset p'$), with $p \neq p'$, p' receives some properties from p, and p' is a sub-role, or specialization, of p. In the definition of the role properties presented in the sequence, it will be precisely stated what one specialized role inherits from another role. For example, in the soccer domain, the attacker role has many properties of the player role ($p_{player} \subset p_{attacker}$). It is also possible to state that a role specialize more than one role, i.e., a role can receive properties from more than one role. The set of all roles are denoted by R_{ss}. Following this OO inspiration, we can define an abstract role as a role that cannot be played by any agent. It has just a specification purpose. The set of all abstract roles is detonated by $R_{abc}(R_{abc} \subset R_{ss})$. There is also a special abstract role $P_{soc}$where $\forall_{(p\in\ R_{ss})} P_{soc} \subset P$, trough the transitivity of $\subset$, all other roles are specializations of it.
- **Social level** is when the inheritance relation does not have a direct effect on the agents' behavior, there are other kinds of relations among roles that directly constrain the agents. Those relations are called links and are represented by the predicate $link_{(p_s,p_d,t)}$ where p_s is the link source, p_d is the link destination, and $t \in acq, com, aut$ is the link type. In case the link type is acq (acquaintance), the agents playing the source role p_s are allowed to have a representation of the agents playing the destination role p_d (p_d agents, in short).In a communication $link(t = com)$, the p_s agents are allowed to communicate with p_d agents. In an authority $link(t = aut)$, the p_s agents are allowed to have authority on p_d agents, i.e., to control them. An authority link implies the existence of a communication link that implies the existence of an acquaintance link:

$$link_{p_s,p_d,aut} \Longrightarrow link_{p_s,p_d,com} \tag{1.1}$$

$$link_{p_s,p_d,com} \Longrightarrow link_{p_s,p_d,acq} \tag{1.2}$$

  Regarding the inheritance relation, the links follow the rules:

$$(link_{(p_s,p_d,t)} \wedge p_s \subset p_{s'}) \Longrightarrow link_{(p_{s'},p_d,t)} \tag{1.3}$$

$$(link_{(p_s,p_d,t)} \wedge p_s \subset p_{d'}) \Longrightarrow link_{(p_s,p_{d'},t)} \tag{1.4}$$

  Take an instance, if the coach role has authority on the player role $link_{(p_{coah},p_{player},aut)}$ and player has a sub-role ($p_{player} \subset p_{attacker}$), by Eq. (4), a coach has also authority on attackers. Moreover, a coach is allowed to communicate with players (by Eq. (1)) and it is allowed to represent the players (by Eq. (2)).

- **Collective level**: the links constrain the agents after they have accepted to play a role. However we should constrain the roles that an agent is allowed to play depending on the roles this agent is currently playing. This compatibility constraint $p_a \bowtie p_b$ states that the agents playing the role are also allowed to play the role p_b (it is a reflexive and transitive relation). As an example, the team leader role is compatible with the back player role $p_{leader} \bowtie p_{back}$. If it is not specified that two roles are compatible, by default they are not. Regarding the inheritance, this relation follows the rule
$$(p_a \bowtie p_b \wedge p_a \neq p_b \wedge p_a \sqsubseteq p') \Longrightarrow (p' \bowtie p_b) \tag{1.5}$$

Hence, there should be a series of rules and relationships defined within the collective level.

The software & Systems Process Engineering meta-model (SPEM) allows the Modelling of software processes using OMG (Object Management Group) standard such as the MOF (meta-object facility) and UML: making possible to represent software processes using tools compliant with UML. Daniel Rodriguez et al [14] represents generic processes modeled with SPEM using an underlying ontology based on the OWL representation together with data derived from actual projects. SPEM is generally used to design generic software processes such as Scrum. Therefore, Daniel Rodriguez et al [14] discussed a first approach to create an ontology from the Scrum process as example using SPEM. In Daniel Rodriguez et al [14], Scrum Ontology extends from the *Role* class in the method-content ontology in SPEM. *Method-Content: Role* has *scrum:Pig* and *scrum:Chicken* as subclasses. The *productOwner*, *scrumMaster* and *team* were instances of the *Scrum Chicken role*. Moreover *WorkProduct* as one class in the method content ontology has three subclasses: *Artifact*, *Deliverable* and *Outcome*. The term like SprintBacklog is as an instance or individual of *Artifact* as the part of the scrum Ontology. Furthermore, the process ontology of SPEM has *Activity* class with *Iteration* and *Phase* classes defined as subclasses. Relatively, the *PreGame*, *Game* and *PostGame* are defined in the scrum ontology.

One of the earliest initiatives was the TOVE project that aimed at development of a set of integrated ontologies for modeling all kinds of enterprise, such as commercial and public ones. The TOVE Organization Ontology [8] for Enterprise Modelling is one of these, which puts forward a number of conceptualizations as agents, roles, positions, goals, communication, authority, commitment. Precisely, one organization consists a set of Organization-Agents(OA) having two sub-classes: Individual-Agents and Group-Agents, a set of Organization-Units as recursive subcomponents and an Organization-Goal tree (could divide into sub-goals). An Organization-Role defines a prototypical function of an agent in an organization. Each Organization-Role played by OA has Organization-Goals or Role-Goals, Role-Skills, Role-Process or Organization-Activity, Role-Policy, Role-Communication-Link. Moreover, an Organization-Position defines a formal position that can be filled by OA in the Organization.

1.5 Conclusion

The goal of this paper was to present K-CRIO, an ontology of organizations for their understanding, analysis and also to enable reasoning. The targeted organizations are those dedicated to the realization of products following a given design process. The definition of this ontology relies on OWL and on the concepts used in the CRIO metamodel. The Scrum software development process is taken as example to illustrate the use of K-CRIO for describing a specific organization.

We are aware that the K-CRIO ontology is not complete. It is a first attempt to build a rich ontology for organizations. Further studies, done with heterogeneous organizations will help us in refining K-CRIO and adding new concepts and relationships. In the meanwhile, the idea is to use K-CRIO as a semantic layer for collaborative softwares in order to enhance Knowledge Management within the targeted organizations.

References

1. http://en.wikipedia.org/wiki/scrum.
2. Stefano Beco, Barbara Cantalupo, Ludovico Giammarino, Nikolaos Matskanis, and Mike Surridge. OWL-WS: A workflow ontology for dynamic grid service composition. In *eScience*, pages 148–155. IEEE Computer Society, 2005.
3. E Bottazzi and R Ferrario. Preliminaries to a dolce ontology of organizations. *International Journal of Business Process Integration and Management*, 4(4):225–238, 2009.
4. Ivan Cantador and Pablo Castells. Multilayered semantic social network modelling by ontology-based user profiles clustering: Application to collaborative filtering. *Springer Verlag Lectures Notes in Artificial Intelligence*, pages 334–349, 2006.
5. Massimo Cossentino, Nicolas Gaud, Stéphane Galland, Vincent Hilaire, and Abderrafiaa Koukam. A holonic metamodel for agent-oriented analysis and design. In *HoloMAS'07*, pages 237–246, 2007.
6. Massimo Cossentino, Nicolas Gaud, Vincent Hilaire, Stephane Galland, and Abderrafaa Koukam. Aspecs: an agent-oriented software process for engineering complex systems. *Autonomous Agents and Multi-Agent Systems*, 20:260–304, 2010. 10.1007/s10458-009-9099-4.
7. Pete Deemer, Gabrielle Benefield, Craig Larman, and Bas Vodde. The scrum primer. Technical report, www.goodagile.com, 2010.
8. Mark S. Fox, Mihai Barbuceanu, and Michael Gruninger. An organisation ontology for enterprise modeling: Preliminary concepts for linking structure and behaviour. *Computers in Industry*, 29(1-2):123 – 134, 1996. WET ICE '95.

9. Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. A model for the structural, functional, and deontic specification of organizations in multiagent systems. In *Proceedings of the 16th Brazilian Symposium on Artificial Intelligence: Advances in Artificial Intelligence*, SBIA '02, pages 118–128, London, UK, UK, 2002. Springer-Verlag.
10. Yishuai Lin, Vincent Hilaire, Nicolas Gaud, and Abderrafiaa Koukam. K-crio: an ontology for organizations involved in product design. In *Proceedings of the DICTAP'11 conference*, number 167 in Communications in Computer and Information Science series. Springer, 2011.
11. David Martin, Mark Burstein, Drew McDermott, Sheila McIlraith, Massimo Paolucci, Katia Sycara, Deborah L. McGuinness, Evren Sirin, and Naveen Srinivasan. Bringing semantics to web services with OWL-S. In *First International Workshop on Semantic Web Services and Web Process Composition*, pages 243–277, 2004.
12. Deborah L. McGuinness and Frank Van Harmelen. http://www.w3.org/tr/owl-features/.
13. Peter Mika. Ontologies are us: A unified model of social networks and semantics. *J. Web Sem*, 5(1):5–15, 2007.
14. Daniel Rodrguez-Garca, Elena Garca Barriocanal, Salvador Snchez Alonso, and Carlos Rodrguez-Solano Nuzzi. Defining software process model constraints with rules using owl and swrl. *International Journal of Software Engineering and Knowledge Engineering.*
15. Leo Sauermann, Ludger Van Elst, and Andreas Dengel. A.: Pimo - a framework for representing personal information models. In *In Proceedings of I-Semantics 07, JUCS*, pages 270–277, 2007.
16. Guus Schreiber. http://www.cs.vu.nl/ guus/public/owl-restrictions/.
17. ScrumMethodology.org. Scrum phases, http://www.scrummethodology.org/scrum-phases.html, 2009.
18. David Vallet, Phivos Mylonas, Miguel A. Corella, Jos M. Fuentesa, Pablo Castells, and Yannis Avrithis. A semantically-enhanced personalization framework for knowledge-driven media services. In *IADIS WWW/Internet Conference*, pages 11–18, 2005.