

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | Florian Zysset  Leeroy Brun |

Gestion de tâche en XML

Table des matières

1. [Introduction 2](#_Toc401497732)
2. [Cahier des charges 3](#_Toc401497733)

[Tâche](#_Toc401497734)

[Personne](#_Toc401497735)

[Lieu](#_Toc401497736)

[Evénement/Condition](#_Toc401497737)

[UI](#_Toc401497738)

1. [Modélisation 7](#_Toc401497739)

[Tasks](#_Toc401497740)

[Resources](#_Toc401497741)

[Locations](#_Toc401497742)

[Events](#_Toc401497743)

[UI](#_Toc401497744)

1. [Document Type Definition 10](#_Toc401497745)

[Tasks](#_Toc401497746)

[Resources](#_Toc401497747)

[Locations](#_Toc401497748)

[Events](#_Toc401497749)

[UI](#_Toc401497750)

1. [Schéma XML 13](#_Toc401497751)

[Tasks](#_Toc401497752)

[Resources](#_Toc401497753)

[Locations](#_Toc401497754)

[Events](#_Toc401497755)

[UI](#_Toc401497756)

1. [Conclusion 16](#_Toc401497757)

[Fonctionnalités intéressantes mais indisponible](#_Toc401497758)

[Point intéressant de notre modélisation](#_Toc401497759)

[En conclusion](#_Toc401497760)

1. [Annexe 16](#_Toc401497761)

# Introduction

Dans le cadre de notre formation à la HEIG-VD, et durant le module “AppliServ”, il nous a été demandé de modéliser en XML un utilitaire de gestion de tâches.

Nous avons ensuite dû y ajouter des feuilles de style xsl afin d’afficher élégament les informations de ce fichier XML.

Ce document décrit ce que nous avons choisi d’afficher, pourquoi et comment nous l’avons fait.

# Cahier des charges

Le but est de permettre l’affichage facile des données contenues dans le fichier XML, ce qui inclus :

* Liste de toutes les tâches
* Liste des tâches pour une personne donnée
* Détails d’une tâche donnée
* Diagramme des tâches ainsi que des relations entre elles

Nous allons décrire maintenant chacun de ces aspects.

## Liste des tâches

La liste des tâches permet d’avoir une vue d’ensemble du projet en cours.

Il est important d’y afficher les informations de chaque tâche de façon visuelle :

* Resource assignée à la tâche
* Progrès : affichage d’une barre de progression
* Emplacement
* Durée
* Statut : affiché avec des couleurs reconnaissables, permet d’avoir rapidement une vue d’ensemble

Il est important de pouvoir accéder rapidement au détail de chaque tâche en cliquant sur le titre de celle-ci.

Nous devons aussi pouvoir afficher les tâches assignées à une personne en cliquant sur son nom.

Ceci étant le point d’entrée de notre « application », nous devons également avoir des liens vers :

* Le diagramme des tâches
* Le filtre des tâches pour chaque personne

## Liste des tâches pour une personne donnée

Cette page doit permettre de sélectionner une personne et d’en afficher les tâches auxquelles elle est associée.

Cela permet d’avoir rapidement une vue d’ensemble du travail en cours pour une personne sur un projet.

La liste des tâches d’une personne donnée sera présentée de la même manière que la liste des tâches principale.

## Détails d’une tâche donnée

Cette page permet d’afficher les détails d’une tâche donnée.

Les informations qui seront affichées sont les mêmes que pour la liste des tâches, ainsi que les commentaires associés à la tâche.

## Diagramme des tâches ainsi que des relations entre elles

Il s’agit d’une illustration SVG représentant les différentes tâches du projet, ainsi que des relations qu’elles ont entre elles (les events).

Il est important d’afficher sur chaque relation les conditions de l’événement, afin de savoir ce qui lie chacune des tâches.

Il doit être possible de cliquer sur une tâche pour en afficher les détails.

Afin de gérer les tâches dans le cadre d’un logiciel de gestion de projet, il est nécessaire de décrire la structure de donnée, obligatoirement en XML, permettant de gérer les différentes fonctionnalités définies ci-dessous.

La structure de donnée doit permettre de gérer différent type d’entités différentes :

* Tâche
* Personne
* Lieu
* Evénement
* UI

En effet, une tâche est assignée à une personne, et peut être lié un lieu précis.

Ces tâches ont des relations entre elle. Par exemple, une tâche peut uniquement commencer lorsque la précédente est terminée, ou lorsque la tâche précédente à un avancement d’au moins « 80% ». Ce cas est gérer par les événements.

Pour finir, ces tâches doivent pouvoir être rechargées, et garder le même aspect « Graphique » qu’auparavant. Admettons l’utilisation d’une application HTML permettant de gérer ces tâches, il est nécessaire de pouvoir sauvegarder la position et la taille de chaque tâche.

Les lieux et les localités sont définissables dans l’application. Les conditions peuvent être définies sur chaque « lien » qui peut être créé.

Pour chaque entité, une certaine quantité d’informations décrivant celle-ci sont définis dans le chapitre correspondant.

|  |  |
| --- | --- |
|  | Les attributs en gras sont obligatoires, tandis que les autres sont facultatifs. |

## Tâche

Une tâche est décrite par les informations suivantes :

|  |  |  |
| --- | --- | --- |
| Nom de l’information | Type de données | Remarques |
| **Identification** | Peu importe | Utile pour le suivit et les relations |
| **Titre** | Texte | Exemple : « Gestion de tâche », « Suppression du cours XML » |
| **Statut** | Enumérée | Les valeurs permises sont :   * NOT\_STARTED * IN\_PROGRESS * COMPLETED |
| **Assignation** | Personne | Assignation à une personne |
| Commentaires | Texte libre | Commentaires pouvant être ajouté par les personnes assignées.  Plusieurs commentaires peuvent être insérer par tâche. |
| Progression | Nombre en 0 et 100 | Un pourcentage d’avancement de la tâche estimée par les personnes assignées |
| Location | Lieu | Un lieu où cette tâche doit être réalisée. |
| Durée | Number | Durée de la tâche en heure |
| Priorité | Enumérée | Les valeurs permises sont :   * HIGH * NORMAL * LOW |

## Personne

Une personne est décrite par les informations suivantes :

|  |  |  |
| --- | --- | --- |
| Nom de l’information | Type de données | Remarques |
| **Identification** | Peu importe | Utile pour le suivit et les relations |
| **Titre** | Enuméré | Les valeurs permises sont :   * Monsieur * Madame * Docteur |
| **Nom** | Texte | Le nom de la personne |
| **Prénom** | Texte | Le prénom de la personne |
| Date de naissance | Date | La date de naissance de la personne |
| Biographie | Texte libre | La biographie de la personne |

## Lieu

Un lieu est décrit par les informations suivantes :

|  |  |  |
| --- | --- | --- |
| Nom de l’information | Type de données | Remarques |
| **Identification** | Peu importe | Utile pour le suivit et les relations |
| **Nom** | Texte | Le nom du lieu |

## Evénement/Condition

Une tâche peut, en effet, commencer que lorsque certaines conditions sont remplies au préalable. Par exemple, lorsque la tâche précédente est terminée, ou lorsque la tâche précédente est avancée à plus de 50%.

|  |  |  |
| --- | --- | --- |
| Nom de l’information | Type de données | Remarques |
| **Tâche de début** | Tâche | Tâche depuis laquelle la condition démarre |
| **Tâche de fin** | Tâche | Tâche à laquelle la condition se termine |
| **Liste d’opérateur et d’opérande** | Opérateur et opérande | Liste des opérateurs et opérande permettant de gérer la condition. Ces opérateur/opérande sont écrit afin d’être résolu en notation « Polonaise inverse » |

### Opérande

Le modèle permettant la récupération de valeur d’une tâche est le suivant :

|  |  |  |
| --- | --- | --- |
| Nom de l’information | Type de données | Remarques |
| Tache | Tâche | Tâche sur laquelle l’opérande doit déterminer sa valeur |
| Elément | Texte | Nom du champ sur laquelle la valeur doit être trouvée. |
| Value | Texte | Valeur libre |
| Now | Date | Date du jour |

### Opérateur

Les opérateurs suivant doivent être utilisable :

* Egal
* Pas égal
* Plus grand
* Plus petit
* Et
* Ou

## UI

Les différents éléments doivent pouvoir être affiché à l’écran, de la même manière qu’ils étaient disposés précédemment dans l’affichage.

Chaque tâche est un élément affichable, mais afin de faciliter la gestion hors du domaine et de faciliter l’intégration avec d’autres applications, il est nécessaire de séparer ces valeurs des valeurs d’une tâche normale.

### Type d’élément UI

2 types d’éléments UI sont disponibles :

* Rond
* Rectangle

### Rond

|  |  |  |
| --- | --- | --- |
| Nom de l’information | Type de données | Remarques |
| **Rayon** | Nombre | Nombre représentant le rayon du cercle |
| **Position** | Position | Position à l’affichage |

### Rectangle

|  |  |  |
| --- | --- | --- |
| Nom de l’information | Type de données | Remarques |
| **Taille** | Taille | Taille du rectangle |
| **Position** | Position | Position à l’affichage |

### Position

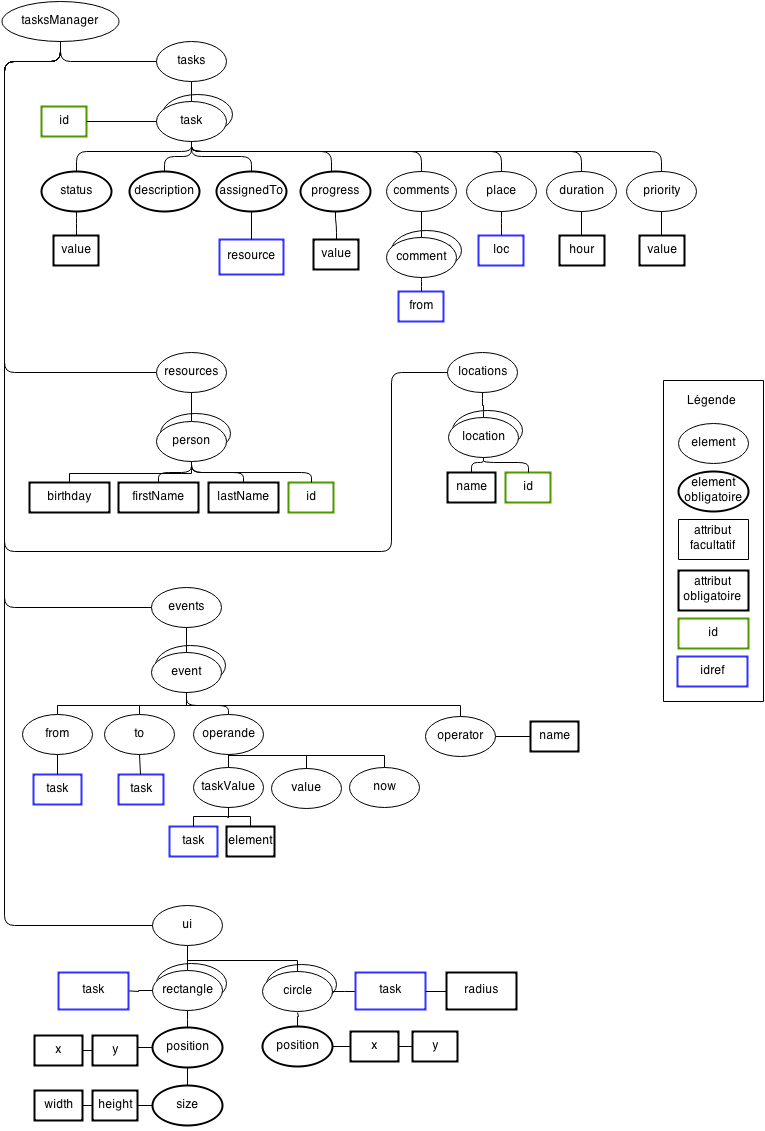
|  |  |  |
| --- | --- | --- |
| Nom de l’information | Type de données | Remarques |
| **X** | Number | Position en X à l’affichage |
| **Y** | Number | Position en Y à l’affichage |

### Taille

|  |  |  |
| --- | --- | --- |
| Nom de l’information | Type de données | Remarques |
| **Width** | Number | Largeur à l’affichage |
| **Height** | Number | Longueur à l’affichage |

# Modélisation

Voici la représentation graphique de notre schéma XML.



La légende présente les différents types d’éléments : nœuds obligatoires, nœuds facultatifs, attributs obligatoires, attributs facultatifs, attributs de type ID, attributs de type IDREF.

Nous avons traduit tous les termes en anglais afin d’être universel. Cela permet de facilement étendre le projet à plusieurs pays, développeurs, etc.

Tous les nœuds sont regroupés sous un noeud principal « tasksManager » (root).

En dessous du nœud principal, tous les types d'éléments sont ensuite regroupés comme suit :

* tasks : toutes les tâches
* resources : toutes les personnes qui peuvent être liées à des tâches
* locations : tous les lieux liés à des tâches
* events : tous les événements qui définissent quand nous passons d’une tâche à une autre

## Tasks

Toutes les tâches sont regroupées ici avec leur noeud spécifique.

Nous pouvons les lier à une ressource et à un lieu à l’aide des noeuds “assignedTo” et “place”. Les ressources peuzvent également laisser des commentaires sur la tache en cours à l’aide du noeud “comments”. l’ID de la tache permet de la lier depuis d’autres éléments (UI et events par exemple).

## Resources

Les ressources (personnes) qui peuvent être liées à des tâches sont regroupées ici.

Nous pouvons les lier ensuite à une tâche grâce à leur “ID”.

## Locations

Les lieux qui peuvent être liées à des tâches sont regroupées ici.

Nous pouvons les lier ensuite à une tâche grâce à leur “ID”.

## Events

Les événements qui définissent le passage d’une tâche à une autre sont définis ici.

Un événement peut avoir une tache de début (from) et/ou une tache de fin (to) (il doit en avoir au moins une des deux).

Il est ensuite possible de définir des opérateurs et opérandes qui définiront les conditions de l’événement (en notation polonaise inverse).

Un opérateur a simplement un nom :

* AND
* OR
* EQUAL
* NOT\_EQUAL
* LESS\_THAN
* BIGGER\_THAN

Une opérande peut soit contenir :

* une valeur
* un noeud “now” qui indiquera qu’il s’agit de la date et heure actuelle
* un noeud “taskValue” qui ira chercher une valeur dans les éléments de la tache

## UI

La représentation graphique des taches a volontairement été séparée des taches en elle-même car des informations n’en font pas vraiment partie. Nous les avons donc regroupées dans un sous-arbre “UI” qui représente toute la partie graphique.

Une tache peut soit être représentée par un rectangle ou un rond qui ont chacun une position (x/y).

Pour le rond, il suffit d’indiquer le rayon dans un attribut “radius”. Concernant le rectangle, nous devons ajouter un noeud “size” avec des attributs “width” et “height”.

# Document Type Definition

Notre schéma DTD s’appelle « task ». La version complète est disponible en annexe de ce document, comme mentionné en fin de document. Cette partie expliquera pas chaque élément du schéma, mais les éléments « spéciaux ».

Comme démontré dans la modélisatio, le nœud « root » défini est « taskManager », et celui-ci contient une liste d’élément possible permettant de sauvegarder les différentes données mentionnées dans le cahier des charges.

<!ELEMENT taskManager (tasks,events,resources,locations,ui)>

Cet élément indique qu’il est nécessaire d’avoir tous les nœuds enfants défini, c’est-à-dire « Tasks », « Events », « Resources », « Locations », « UI ».

La virgule indique une condition « et », donc tous les nœuds sont obligatoires.

## Tasks

Le nœud Tasks est un nœud contenant toutes les tâches (Task). Ce nœud peut en contenir entre 0 et N.

<!ELEMENT tasks (task)\*>

### Task

L’élément enfant est « Task », qui permet de définir les informations pour chaque tâche.

<!ELEMENT task (status,description,assignedTo,progress?,comments?,place?,priority?,duration?)>

<!ATTLIST task

id ID #REQUIRED>

Cet exemple montre que les éléments « status », « description », « assignedTo » sont obligatoires (et doivent être dans le même ordre dans le XML). Cependant, les éléments « progress »,  « comments », « place », « priority » et « duration » sont optionnels.

L’attribut ID (obligatoire) va permettre de faire les relations depuis les autres types de données.

### Status

Voici la déclaration de l’élément « Status », qui est de type énuméré :

<!ELEMENT status EMPTY>

<!ATTLIST status

value (NOT\_STARTED|COMPLETED|IN\_PROGRESS) #REQUIRED>

Ici, la valeur de l’attribut « value » du nœud « status » devra existé dans la liste « NOT\_STARTED|COMPLETED|IN\_PROGRESS ».

### Comments

L’élément « Comments » contient la liste des commentaires.

<!ELEMENT comments (comment)\*>

<!ELEMENT comment (#PCDATA)>

<!ATTLIST comment

from IDREF #REQUIRED>

Chacun des éléments « Comment » est lié à une personne (grâce à un IDREF pour l’attribut « FROM »), et contient un texte libre (PCDATA) comme valeur.

## Resources

L’élément « Resources » contient une liste de personne pouvant être utilisées pour les commentaires et les assignations des tâches.

<!ELEMENT person (#PCDATA)>

<!ATTLIST person

birthday CDATA #IMPLIED

firstName CDATA #REQUIRED

id ID #REQUIRED

lastName CDATA #REQUIRED>

Les noeuds “Person” contiennet une liste d’attributs permettant de décrire cette personne, et le nœud peut contenir du texte servant de biographie de cette personne.

## Locations

<!ELEMENT location EMPTY>

<!ATTLIST location

id CDATA #REQUIRED

name CDATA #REQUIRED>

Dans notre application, l’élément « location » est uniquement décrit par son nom.

## Events

Le nœud « Events” contient les événements et conditions de passage à d’une tâche à une autre.

Elle contient donc des « Event » décrit par la définition suivante :

<!ELEMENT event (((from,to)|(from)|(to)), (operande|operator)+)>

<!ELEMENT operande (taskValue|now|value)>

…

<!ATTLIST operator

name (AND|OR|EQUAL|NOT\_EQUAL|LESS\_THAN|BIGGER\_THAN) #REQUIRED>

…

<!ELEMENT from EMPTY>

<!ATTLIST from

task IDREFS #REQUIRED>

L’élément peut contenir soit un « From », un « to », ou les 2. Mais une de ces combinaisons est obligatoire. Ensuite, le nœud contient des « operande » et des « operator ».

Un opérande peut être un des 3 type de nœuds définis (TaskValue, now, et value), qui représente :

* TaskValue 🡪 Une valeur d’une des tâches, par exemple son statut
* Now 🡪 La date du jour
* Value 🡪 Une valeur fixe

Un opérateur lui contient uniquement un « nom », qui sera utilisé pour réaliser la condition/l’opération par le programme.

Afin de permettre la liaison d’un événement avec plusieurs tâches, nous avons utilisé « IDREFS », qui permet de décrire une liste d’ID.

## UI

La partie UI permet de gérer l’affichage du gestionnaire des tâches. Cela permet de sauvegarder la position et la taille des différents éléments.

<!ELEMENT ui (rectangle|circle)\*>

<!ELEMENT rectangle (position,size)>

<!ATTLIST rectangle

task IDREF #REQUIRED>

<!ELEMENT position EMPTY>

<!ATTLIST position

x CDATA #REQUIRED

y CDATA #REQUIRED>

<!ELEMENT size EMPTY>

<!ATTLIST size

height CDATA #REQUIRED

width CDATA #REQUIRED>

<!ELEMENT circle (position)>

<!ATTLIST circle

task IDREF #REQUIRED

radius CDATA #REQUIRED>

Cette liste contient de rectangles ou des cercles, qui sont liés à une tâche. Chacun de ces 2 éléments contient aussi un nœud « position », permettant d’’indiquer la position de l’élément en X et en Y.

De plus, l’élément rectangle doit posséder un nœud « Size » permettant de gérer sa hauteur et sa largeur.

Le cercle lui à un attribut « radius » permettant de décrire le rayon de ce cercle.

# Schéma XML

## Tasks

Voici un exemple d’une liste de tâche, contenant une seule tâche.

<tasks>

<task id="task\_0">

<status value="IN\_PROGRESS"/>

<description>Create a task manager</description>

<assignedTo resource="res\_1"/>

<progress value="80"/>

<comments>

<comment from="res\_1">Started to work on this ! :D</comment>

</comments>

<place loc="loc\_2"/>

<duration hour="200"/>

</task>

</tasks>

Cette tâche possède un ID « task\_0 » qui permet de la lier à d’autres éléments. Elle est en statut « IN\_PROGRESS », qui indique que cette tâche est en cours de réalisation.

Elle est assignée à « res\_1 », qui est une ressource décrite dans l’arbre « Resources » de ce document XML.

Elle possède un lieu lié, « loc\_2 », ainsi qu’une durée de 200h.

## Resources

<resources>

<person firstName="Florian" lastName="Zysset" birthday="1990-12-02" id="res\_1">

Florian Zysset travaille à Infoteam etc...etc...

</person>

<person firstName="Leeroy" lastName="Brun" birthday="1992-02-19" id="res\_2"/>

</resources>

L’arbre « Resources » contient 2 personnes qui peuvent être liées à des tâches ou des commentaires.

La ressource « res\_1 » est en plus décrite avec l’aide d’un biographie.

## Locations

<locations>

<location name="Cuisine" id="loc\_1"/>

<location name="HEIG-VD" id="loc\_2"/>

<location name="Home" id="loc\_3"/>

</locations>

La liste de location contient 2 locations, qui peuvent être utilisé pour les liaisons avec les tâches.

## Events

<events>

<event>

<from task="task\_1"/>

<to task="task\_2"/>

<operande>

<taskValue task="task\_1" element="status"/>

</operande>

<operande>

<value>COMPLETED</value>

</operande>

<operator name="EQUAL"/>

<operande>

<taskValue task="task\_1" element="progress"/>

</operande>

<operande>

<value>70</value>

</operande>

<operator name="BIGGER\_THAN"/>

<operator name="OR"/>

</event>

</events>

Cet événement, qui part de la tâche « task\_1 », jusqu’à la tâche « task\_2 »

Afin de passer à la task 2, depuis la task 1, il faut qu’une des 2 conditions sont remplies (« OR »). La première condition est que la task\_1 doit avoir le statut « Completed ». La second condition veut que la tâche soit avec un avancement (« Progress » de 70 % ou plus.

<event>

<to task="task\_0 task\_1 task\_2"/>

<operande>

<now/>

</operande>

<operande>

<value>2014-10-20</value>

</operande>

<operator name="BIGGER\_THAN"/>

</event>

Ce 2ème exemple comment réaliser une « date de début » pour une tâche. En effet, cet événement décrit que pour aller aux tâches « TASK\_0 », « TASK\_1 » et « TASK\_2 » il faut que la date du jour (« Now ») soit plus grande que 2014-10-20. Il sera donc impossible de débuter ces tâches avant cette date du 20.10.2014.

<event>

<to task="task\_0 task\_1 task\_2"/>

<operande>

<now/>

</operande>

<operande>

<value>2014-10-20</value>

</operande>

<operator name="BIGGER\_THAN"/>

</event>

Cet événement semblable, hormis que le nœud utilisé est « To », indique les tâches doivent se terminer au maximum le 20.10.2014.

## UI

<ui>

<rectangle task="task\_1">

<position x="1" y="2"/>

<size height="200" width="200"/>

</rectangle>

<circle task="task\_2" radius="100">

<position x="100" y="350"/>

</circle>

</ui>

Dans cet exemple, la tâche « task\_1 » est décrite comme un rectangle à l’affichage, positionné en X = 1, Y = 2, avec une taille de 200 sur 200.

La tâche « task\_2 » est un cercle, avec un radius de 100. Il est positionné en X=100 et Y=350.

# Conclusion

Le schéma que nous avons développé permet de gérer les principales fonctionnalités d’un logiciel permettant la gestion de Projet. Cependant, dût au temps limité à notre disposition, et un cahier des charges que nous avons défini nous-même, certaines fonctionnalité qui auraient été intéressantes ont été mises de côtés.

## Fonctionnalités intéressantes mais indisponible

Dans les fonctionnalités intéressantes nous pouvons citer la gestion des tâches « Parent-Enfant ». En effet, dans le cadre d’une méthodologie de projet « Scrum » ou « Agile », une tâche peut avoir des relations « parent-enfants ». Par exemple, un « User Story » serait une tâche parente, avec comme tâche enfant les différentes étapes de son implémentation.

Une autre fonctionnalité serait la gestion du temps de réalisation, avec une fonctionnalité de « Tracking » du temps passé sur une tâche, afin de calculer automatiquement l’avancement d’une tâche.

Un dernier ajout qui serait intéressant serait la gestion d’une « Workflow » pour une tâche. En effet, une tâche doit, normalement, avant d’atteindre le statut « Completed » passé par le statut « In Progress ». Tout comme il ne pourrait normalement pas être créé directement avec le statut « Completed ». Ces workflows pourraient, avec la fonctionnalité « 1 » de ce chapitre, être lié à un type de tâche. (« User Story », « Tasks », « Bug »,…)

## Point intéressant de notre modélisation

Certaines validations que nous avons mis en place, tel que pour les « event » où il est nécessaire d’avoir « FROM », « TO », ou « FROM-TO » est un point vraiment positif qui ne permettrait pas d’avoir des événements incomplets/incohérents.

La gestion des UI/Resources/Location dans des sous arbres des tâches, et pas directement inclus dans les tâches, permettrait la gestion de ces informations dans des fichiers externes au fichier de Task. Ce point nous parait très intéressant dans le cadre d’une intégration avec d’autres systèmes, par exemple un système de gestion des ressources. Il serait dès lors possible de gérer les personnes disponibles grâce à cet autre système, et généré uniquement le fichier des ressources disponibles.

## En conclusion

Nous sommes contents de notre implémentation, qui permet une évolution intéressante dans le cas d’une application réelle de ce schéma. Nous restons cependant convaincus que certaines fonctionnalités décrites ci-dessus apporteraient une réelle « Plus-Value » à notre produit.

# Annexe

* Task.dtd 🡨 Document Type Definition
* Tasks.xml 🡨 Schema XML validé avec task.dtd. Utilisé comme exemple pour valider la structure du DTD.