UdG Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Enginyeria Informàtica

Títol: Llenguatge d'assemblatge per a la construcció i simulació de mecanismes senzills

Document: Memòria

Alumne: Aura Ruiz Duñach

Director/tutor: Gustavo Patow

Departament: Informàtica, Matemàtica Aplicada i Estadística

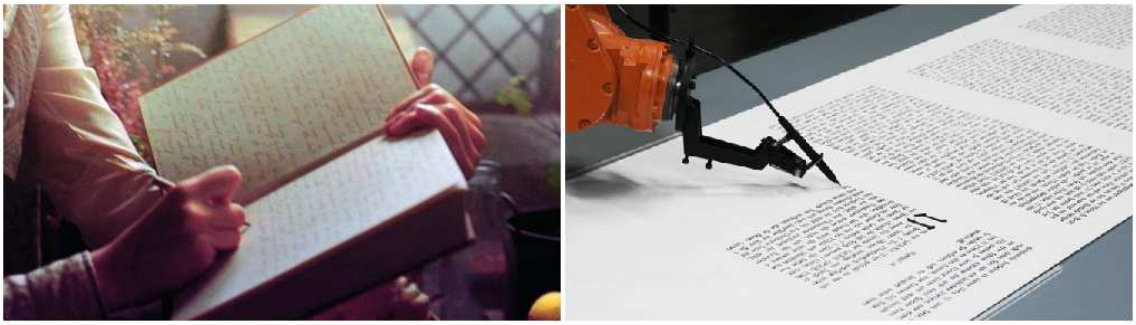
Àrea: LSI

Convocatòria (mes/any): Setembre/2015

Índex

1. Introducció

En els darrers temps, el món tecnológic ha experimentat una evolució molt significativa que segueix en augment. L'objectiu de les noves tecnologies és aconseguir sistemes més intel·ligents, dinàmics i àgils d'utilitzar; eines que pugin estalviar temps i esforç a obtenir els mateixos resultats que sense el seu ús. Tot i que els diversos àmbits d'estudi no es basen en l'ús de la informàtica, l'enginyeria informàtica la podem trobar a qualsevol lloc: cultura, esport, educació, administració, finances, mecànica, construcció,... . No és una excepció el món de la indústria, i la fabricació de maquinària, la qual ha evolucionat considerablement gràcies a l'ús dels sistemes informàtics centrats en aquest àmbit.



1.1 - Exemple d'escriptura manual i robòtica

1.1 Motivacions

Per a la posterior construcció de maquinària, inclús abans de construir físicament un prototip, cal fer un disseny i un estudi d'assemblatge del mecanisme pensat. Avui dia existeixen programes dissenyats per facilitar la construcció i simulació de tot tipus de mecanismes, on es poden unir un conjunt de peces abans de construir, establir el millor ordre d'assemblatge o inclús simular els materials i les forces a què pot estar sutmès. No obstant, la major part del procediment per dissenyar un mecanisme (complex o no) és, actualemnt, a mà: el disseny d'algunes peces individuals, les restriccions per unir les peces, les restriccions de moviment,... fent que la tasca de disseny sigui una feina molt repetitiva i tediosa. És a dir, no existeix cap llenguatge d'instruccions per poder dur a terme aquest procés des d'un punt de vista algorítmic.

1.2 Propòsits

El propòsit d'aquest treball de final de grau és ajudar als dissenyadors d'assemblatges mecànics, reduint el temps que dediquen a l'estudi previ i disseny de cada projectes, a dissenyar i simular els mecanismes desitjats.

L'objectiu principal és dissenyar un procés àgil, que automatitzi l'assemblatge a l'espai entre peces o conjunts de peces, establint posicions relatives entre elles. De manera

senzilla, tal com especificar les relacions entre objectes a partir d'un llenguatge de regles assembladores en forma algorítmica, es vol indicar elements, o conjunts d'elements, a crear i la relació entre aquests per unir-los entre si. A més, amb l'objectiu de ser útil per fer estudis previs a la creació de prototips, es vol donar la opció d'especificar motors dintre de l'assemblatge, amb els quals es podrà simular el moviment del mecanisme.

Aquesta aplicació pot donar peu a construir les bases més simples d'assemblatges més complexos, amb les quals, els usuaris finals podran modificar i personalitzar al gust les estructures, estalviant-se temps en la base.

1.3 Objectius

El projecte es durà a terme sobre una plataforma que permeti dissenyar i simular el comportament d'un conjunt de peces unides, o no, entre si, com és Autodesk Inventor. Les tasques inicials a desenvolupar són:

- Estudi del funcionament d'Autodesk Inventor i la seva API.
- Estudi de la possibilitat d'implementar el projecte amb Python com alternativa al C#. En cas contrari, es continuaria el desenvolupament en C#.
- Adaptació del projecte al nou llenguatge.
- Estudi dels graus de llibertat en la transmissió de moviment dels mecanismes.
- Estudi dels bloquejos de moviment en la simulació automàtica i manual.
- Desenvolupament del llenguatge per a simulacions automàtiques
- Ampliació del llenguatge per a nous components dintre el mecanisme.
- Finalització i arrodoniment de la documentació del treball realitzat.

1.4 Estructura del document

La memòria segueix una estructura de quinze punts que engloben desde l'inici de la idea original del projecte, passant per la planificació, el desenvolupament i la fase de proves, fins arribar a les conclusions finals del mateix.

1- Introducció

Aquest punt situa el marc del projecte i s'exposen les raons que justifiquen el desenvolupament, s'estableixen els objectius inicials que s'esperava obtenir i l'estructura de com s'ha organitzat la documentació.

2- Estudi de viabilitat

Aquest punt recull els paràmetres que han fet possible el desenvolupament del projecte, tal com són els recursos tècnics necessaris, recursos humans, tecnologies emprades i el cost econòmic.

3- Metodologia

Explicació de la metodologia de desenvolupament del projecte escullida i els motius de la tria.

4-Planificació

Aquesta etapa defineix el *timing* del projecte, és a dir, s'exposa el pla de treball, les tasques planificades, el temps estimat i els resultats esperats de cada tasca.

5- Marc de treball i conceptes previs

Introducció als conceptes relacionats amb el desenvolupament del projecte i la seva temàtica, necessaris per aconseguir un bon seguiment del projecte.

6- Requisits del sistema

Descripció dels requisits funcionals i no funcionals del sistema per donar a coneixer tot allò que cal fer perquè el projecte compleixi elsl seus objectius.

7- Estudis i decisions

Descripció del maquinari, les llibreries i/o programes utilitzats per desenvolupar el projece, i el motiu de la seva tria.

8- Anàlisi i disseny del sistema

Explicació de l'anàlisi, que s'encarrega d'estudiar de manera detallada les necessitats del sistema, i el disseny del sistema, que proposa una solució. A la fase d'anàlisi es detallarà el model de dades i el model de processos. A la fase de disseny es determinarà les interfícies d'usuaris, els models físics de dades, els models físics de processos i els models d'objectes que s'hagin utilitzat.

9- Implementació i proves

En aquest apartat es detallen els problemes trobats durant el desenvolupament del projecte i com s'han solucionat.

10- Resultats

Descripció detallada del procés de desenvolupament del projecte i mostra dels resultats obtinguts i del grau d'assoliment dels objectius.

11- Conclusions

Reflexió sobre l'assoliment dels requisits establerts, explicació de les possibles desviacions de la planificació inicial i crítica dels resultats obtinguts.

12- Treball futur

En aquest punt es valora les possibles ampliacions o millores del projecte per un treball futur.

13- Bibliografia

En aquest apartat es mostra les referències utilitzades per desenvolupar el projecte.

14- Annexos

Recull d'informació complementària del projecte.

15- Manual d'usuari i/o instal·lació

En aquest punt s'explica com s'ha d'utilitzar l'aplicació i, si fos necessari, com s'ha d'instal·lar.

2. Estudi de viabilitat

2.1 Recursos tècnics pel desenvolupament del projecte

Els recursos tècnics requerits pel desenvolupament del projecte coincideixen amb els requerits per la plataforma de disseny i simulació que utilitzarem: Autodesk Inventor. Segons la versió que s'utilitzi, variarar els requisits mínims del processador, targeta gràfica, memòria... sempre sobre el sistema operatiu de Windows. Veure annex *14.1 - Requeriments d'Autodesk Inventor*.

Pel projecte s'ha utilitzat un ordinador amb el sistema operatiu Windows 8.1 de 64 bits amb un processador Intel Core i7-4510U, amb una memòria RAM de 16 GB i una targeta gràfica NVIDIA GeForce 840M. Amb aquestes característiques es va haver d'utilitzar Autodesk Inventor 2015.

2.2 Recursos humans

El projecte requeria dels següent rols:

- Analista de sistemes: encarregat del disseny i obtenció dels algoritmes, sempre amb l'objectiu de crear i millorar l'eficiència de l'aplicació, entre altres.
- Cap de projecte: encarregat de conduir el projecte desde la concepció original fins la presentació final.
- Programador: desenvolupador del codi font de la aplicació del projecte, que engloba la funció d'escriure, depurar i mantenir el codi font del programa, seguint els passos definits per l'analista.

Tot i establir-se a la teoria aquest diversos rols, a la pràctica els diversos papers han estat duts a terme per mi, amb la supervisió i ajuda del tutor del projecte en el rol de cap de projecte.

2.3 Tecnologia

Basant-nos en els requisits exposats a l'apartat 2.1, necessitarem algunes eines més per poder desenvolupar i dur a termen el projecte.

Inicialment, com s'ha comentat anteriorment, és necessari instal·lar la plataforma d'*Autodesk Inventor 2015*, que serà la plataforma principal de treball sobre on s'executarà la nostra aplicació. Com a plataforma de desenvolupament s'utilitzarà *Microsoft Visual Studio 2013*, que conté totes les eines necessàries per connectar-se a Inventor i poder desenvolupar amb C# i python.

2.4 Cost econòmic

En aquest apartat detallem el cost econòmic que comporten els recursos tècnics, la tecnologia i els recursos humans dels apartats anteriors.

En el cas dels recursos tècnics, a causa de comprar un ordinador portàtil on dur a terme el projecte, hi ha un cost que equival al cost proporcional d'aquest ordinador per la durada dintre el projecte. El cost d'amortització segueix la següent fòrmula:

$$Amortització = \frac{Preu\ recurs * mesos\ feina}{36\ mesos}$$

L'amortització pel nou ordinador, amb un preu de 960€, i una integració en el projecte de 12 mesos, és de:

$$Amortització = \frac{960 * 12}{36} = 320\ €$$

En referència a la tecnologia, Autodesk proporciona una llicència gratuïta a estudiants i professors per utilitzar algunes de les seves aplicacions, *Inventor 2015* inclós. *Microsoft Visual Studio 2013* proporciona una llicència individual sense cap cost.

Des del punt de vista teòric, els recursos humans tenen diferents costos segons el rol que s'està desenvolupant siguent, com exemple, el cost de l'analista per hora superior al del programador:

- Analista: 20 €/h
- Programador: 10 €/h

Distingim les següents tasques a desenvolupar dintre el projecte:

- Estudi de les eines
- Disseny de l'estructura de classes i algorismes principals
- Implementació
- Proves
- Memòria

Daquesta manera, el cost en funció del perfil i les tasques es mostra a la taula següent:

Tasques	Perfil	Hores	Cost/h	Cost total
Estudi d'eines	Analista	100 h.	20 €/h	2000 €
Disseny classes/algorismes	Analista	100 h.	20 €/h	2000 €
Implementació	Programador	400 h.	10 €/h	4000 €
Proves	Programador	300 h.	10 €/h	3000 €
Memòria	Analista	150 h.	20 €/h	3000 €
Total Programador		*700 h.*	*10 €/h*	*7000 €*
Total Analista		*350 h.*	*20 €/h*	*7000 €*
COST TOTAL		**1050 h.**		**14000 €**

Com s'ha dit, aquests serien els costos teòrics si s'hagués hagut de contractar a dues persones per realitzar els rols esmentats, però, a la pràctica, totes les tasques s'han realitzat sense cap retribució econòmica.

2.5 Conclusions

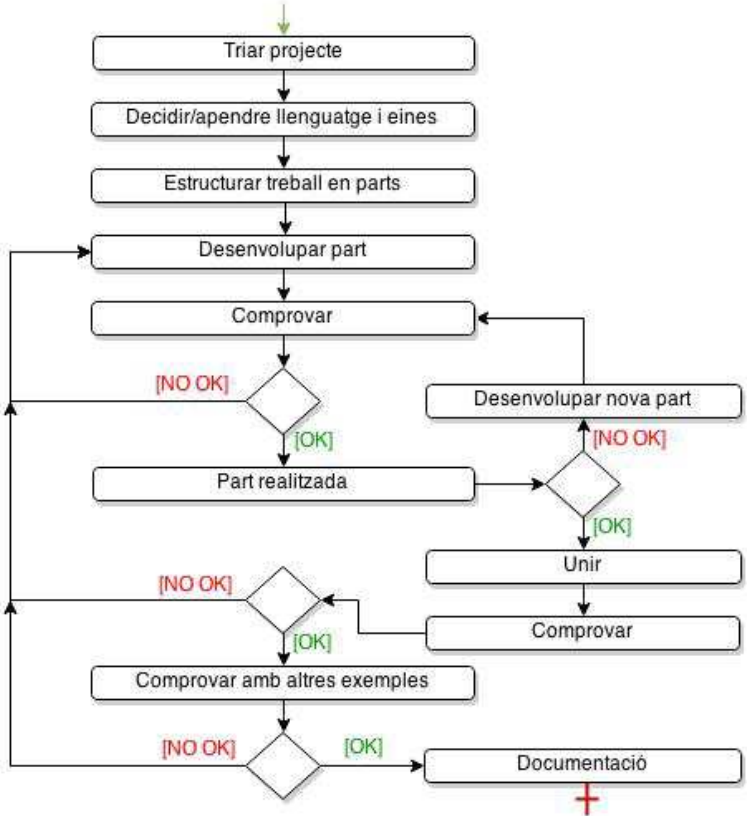
Podem veure pels punts anteriors que la viabilitat del projecte de recerca està garantit a nivell tècnic i tecnològic a priori. És així perquè ja disposem dels requisits mínims per començar el projecte i que el programari que utilitzarem disposa de llicències lliures o per ús educatiu.

A nivell econòmic, si haguéssim hagut de remunerar a treballadors en funció dels perfils esmentats per dur a terme les tasques anteriors, el cost hauria sigut de 14.000 €. Tot i així, aquest no és el nostre cas.

3. Metodologia

A dia d'avui podem distingir moltes metodologies de desenvolupament eficients i diferenciables, desde les més antigues metodologies com la de *Waterfall* fins de les més modernes tècniques *Agile.* La metodologia d'aquest projecte no segueix cap metodologia d'un tipus concret, ja sigui *Spiral, Scrum* o *Iterative*. Hem utilitzat una metodologia pròpia adient als objectius del projecte.

S'ha definit un tipus de metodologia que s'adapti bé al projecte, tal com podem veure al següent diagrama de flux (Figura 3.1).



3.1 - Diagrama de flux per la metodologia SkylineEngine

Com es pot veure a la Figura 3.1, els passos que segueix la metodologia indicada són:

1. Triar el treball a desenvolupar
2. Decidir el llenguatge de programació i les eines a utilitzar
3. Aprendre el llenguatge de programació i el funcionament de les eines escollides
4. Estructurar el treball en parts segons les funcions que s'han de realitzar
5. Desenvolupar la part corresponent seguint l'ordre de l'estructura del treball
6. Fer comprovacions per tal de confirmar que el funcionament és correcte al finalitzar la part

6.1. Si el resultat no és l'esperat, es torna al punt 5 per realitzar els canvis oportuns de la darrera part desenvolupada, o parts anteriors, si fos necessari
6.2. Si el resultat és l'esperat, s'agafa la part següent i es torna al punt 5 per desenvolupar-la. En cas que s'hagin finalitzat les parts amb les comprovacions respectives, s'inicia el punt 7.

7. Unir totes les parts desenvolupades i comprovar que el funcionament és correcte
   7.1. Si el resultat no és l'esperat, es torna al punt 5 per realitzar els canvis oportuns de l'última part desenvolupada, o anteriors, si fos necessari
   7.2. Si el resultat és l'esperat, s'inicia el punt 8
8. Generar diferents models d'exemple per comprovar que el funconament global és el correcte
   8.1. Si el resultat no és l'esperat, es torna al punt 5 per realitzar els canvis oportuns de l'última part desenvolupada, o anteriors, si fos necessari
   8.2. Si el resultat és l'esperat, s'inicia el punt 9
9. Redactar i polir la documentació del projecte

Com podem veure, consisteix en dividir el projecte en mòduls i establir un temps pel seu desenvolupament, per la verificació i per la correcció del projecte. Al llarg del transcurs del projecte, s'estableixen unes pautes de seguiment, normalemnt setmanals, supervisades mitjançant reunions amb el tutor del projecte. El transcus entre reunions pot variar segons l'etapa en que ens trobem. Normalment, tan les etapes inicials com les de documentació no necessiten reunios tan freqüentment.

L'objectiu al finalitzar un mòdul és no haver-lo de retocar en el futur, de manera que garantim en major part que els errors que puguin sorgir formin part de l'últim mòdul en desenvolupament i/o comprovacions, i no de parts anteriors, o, com a mínim, afectin el mínim possible sense comprometre el progrés del projecte.

Pel procés de disseny utilitzarem el llenguatge de modelitat estàndard dins el camp de l'enginyeria del *software*, l'UML (*Unified Modeling Language*). L'UML s'utilitza per definir un sistema, per detallar els seus elements, per documentar i construir. Per aconseguir-ho, l'UML disposa de nombrosos tipus de diagrames que motren diersos aspectes dels elements representats.

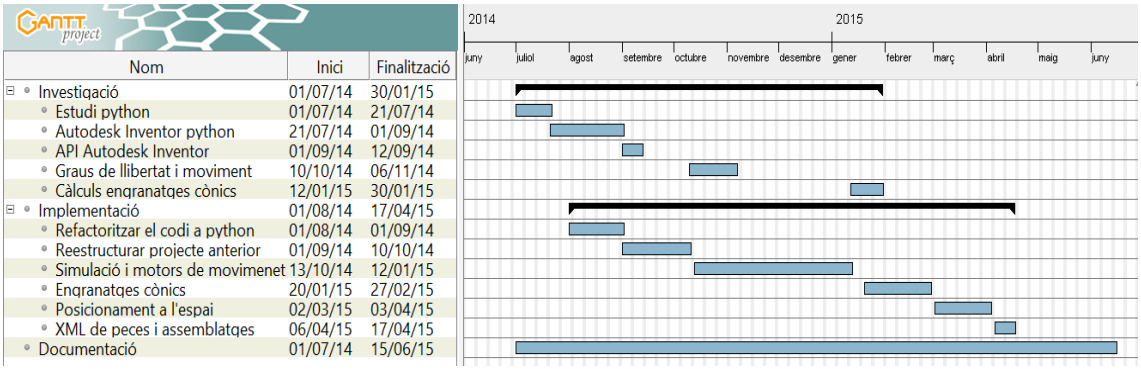
Els motius d'escollir aquesta motodologia han estat per les característiques del projecte mateix. En primer lloc, el projecte es durà a terme per una única persona, de manera que no és necessari detallar amb excés cada part per coordinar-la entre diversos integrants de l'equip. Per tant, aquesta metodologia ens permet prioritzar el procés de desenvolupament en lloc d'invertir molt temps en el disseny previ. Això no treu que s'hagi establert un dissenys abans de començar a desenvolupar.

4. Planificació

Aquest projecte va començar al juliol del 2014 amb la intenció de ser finalitzat i entregat al juny del 2015. Amb aquest propòsit en ment, es van planificar les tasques de la següent manera:

- Juliol 2014 - Setembre 2014. Estudiar el llenguatge python i la manera de com connectar-lo amb l'API d'*Autodesk Inventor 2015*, amb l'objectiu de ralitzar el projecte amb python.
- Inicis de setembre 2014. Establert com a data límit per aconseguir el punt anterior. A partir d'aquí, continuar amb python o retornar al codi C#.
- Setembre 2014 - Octubre 2014. Refer el codi del projecte anterior en mòduls i millorar els processos per carregar els models de peces de l'assemblatge i redefinir les seves restriccions.
- Novembre 2014 - Gener 2015. Estudi d'eines d'*Autodesk Inventor* per a la simulació i moviment dels assemblatges generats. Estudi dels graus de llibertat dels assamblatges. Proves de simulació i automatització del processos necessaris per dur-ho a terme.
- Gener 2015 - Febrer 2015. Estudi dels càlculs per al disseny d'engranatges cònics. Afegir el nou element d'assemblatge.
- Març 2015 - Abril 2015. Afegir concepte d'eix auxiliar de posicionament i angle de posicionament. Redefinir connexions entre elements per posicionar lliurament els elements a l'espai.
- Abril 2015. Generalitzar el proces de lectura de fitxers XML amb un patró composite per poder carregar peces i *assemblies*.
- Abril 2015 - Juny 2015. Arrodoniment de la memòria del projecte.

Cal dir que en el transcurs del projecte, la part d'investigació i la de desenvolupament s'han desenvolupat en paral·lel. Per veure més clarament la planificació del projecte podem observar la Figura 4.1, on es pot veure el diagrama de Gantt.



Nom	Inici	Finalització
Investigació	01/07/14	30/01/15
Estudi python	01/07/14	21/07/14
Autodesk Inventor python	21/07/14	01/09/14
API Autodesk Inventor	01/09/14	12/09/14
Graus de llibertat i moviment	10/10/14	06/11/14
Càlculs engranatges cònics	12/01/15	30/01/15
Implementació	01/08/14	17/04/15
Refactoritzar el codi a python	01/08/14	01/09/14
Reestructurar projecte anterior	01/09/14	10/10/14
Simulació i motors de movimenet	13/10/14	12/01/15
Engranatges cònics	20/01/15	27/02/15
Posicionament a l'espai	02/03/15	03/04/15
XML de peces i assemblatges	06/04/15	17/04/15
Documentació	01/07/14	15/06/15

4.1 - Diagrama de Gantt amb la planificació inicial del projecte

5. Marc de treball i conceptes previs

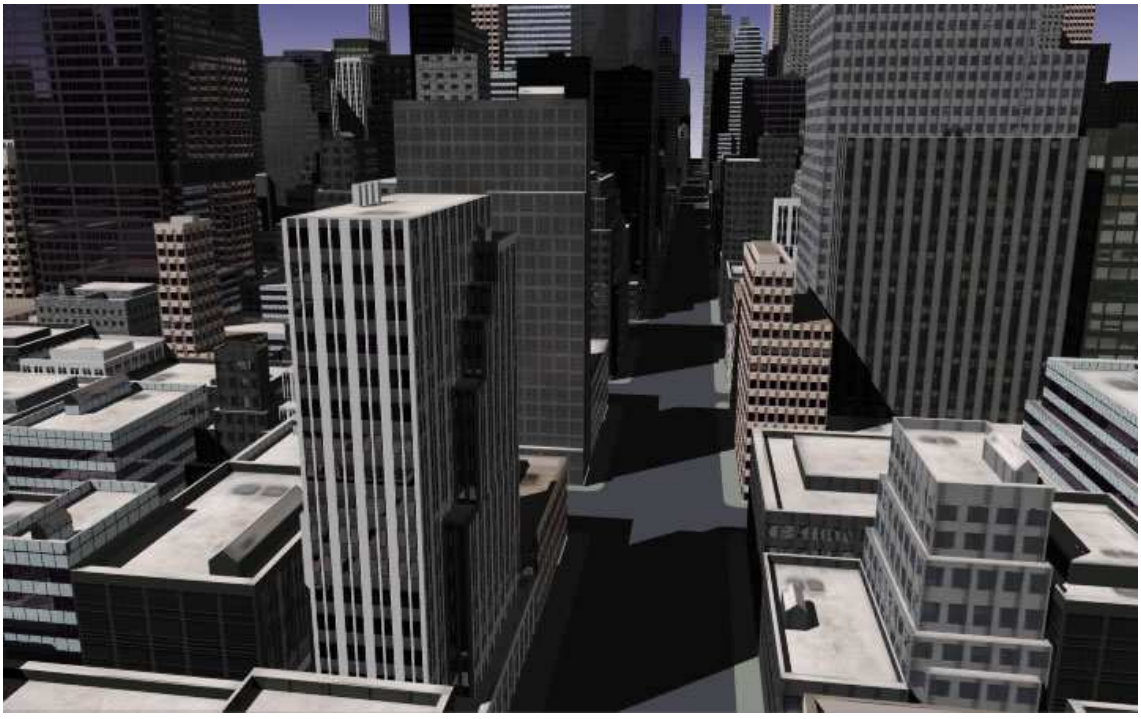
A continuació s'explicarà els conceptes necessaris per poder entendre el projecte i poder-ne fer un bon seguiment. Els conceptes a explicar són:

- Modelatge procedural
- Simulació de maquinària
- Restriccions i graus de llibertat

5.1 Modelatge procedural

El terme de modelatge procedural engloba una sèrie de tècniques de gràfics per computador per crear models 3D i textures a partir d'un conjunt de regles, en lloc de construir a partir de formes geomètriques primitives, com cubs o esferes. Aquest conjunt de regles poden formar part del codi o bé ser independents del motor d'avaluació.

En altres paraules, el modelatge procedural es centra en la generació d'un model basat en un conjunt de regles o restriccions, sense la necessitat d'editar el model a través de l'entrada de l'usuari. Com a conseqüència, l'aplicació de tècniques de modelatge procedural és més eficient que el modelat tradicional. Sovint s'apliquen tècniques de modelatge procedural quan la feina de crear un model 3D utilitzant modeladors és massa feixuga o són necessàries eines més especialitzades. La creació de plantes, arquitectura o paisatges en són exemples (veure Figura 5.1.1).

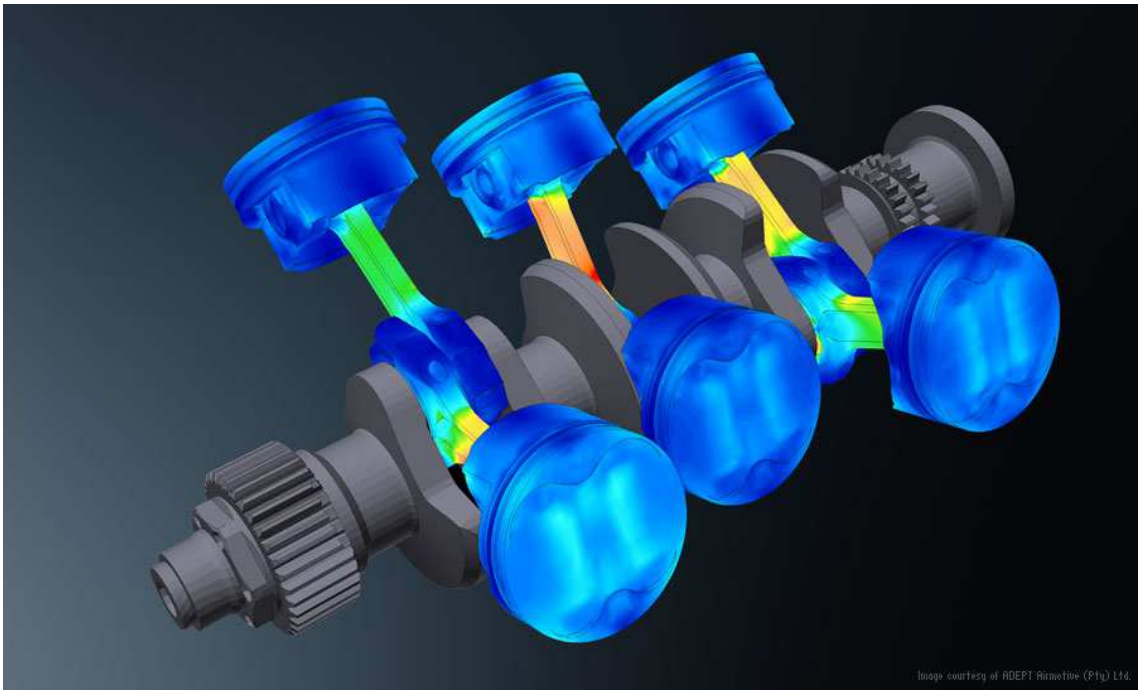


5.1.1 - Exemple de modelatge procedural de ciutats

5.2 Simulació de maquinària

A dia d'avui, els sistemes computacionals estan molt presents en activitats de l'enginyeria moderna. La simulació de maquinària permet aplicar lleis mecàniques sobre un disseny virtual abans de crear un prototip, de manera que poguem estudiar el comportament simulant les condicions de funcionament, igual com poder simular la resistència dels materials a utilitzar o les forces de treball.

La simulació permet estudiar el mecanisme en les seves primeres etapes i comprovar si és possible la construcció o si és necessari realitzar modificacions pel correcte funcionament. D'aquesta manera evitem realitzar diversos prototips d'un mateix mecanisme per realitzar aquest estudi i reduir el cost i el temps d'aquesta etapa.

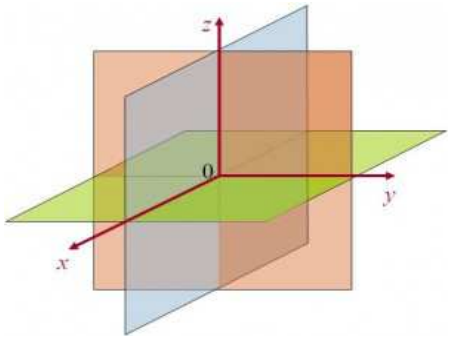


5.2.1 - Exemple de simulació de moviment i forces

5.3 Restriccions i graus de llibertat

En el món del disseny tècnic és impressidible tractar amb sistemes de coordenades i sistemes de referència tant de l'espai com de les peces.

- Sistema de coordenades: conjunt de valors que permeten definir unívocament la posició de qualsevol punt en l'espai respecte a un punt de referència.
- Sistema de referència: Conjunt d'eixos, punts o plans que conflueixen en l'origen i a partir dels quals es calculen les coordenades.



5.3.1 - Sistema de referència

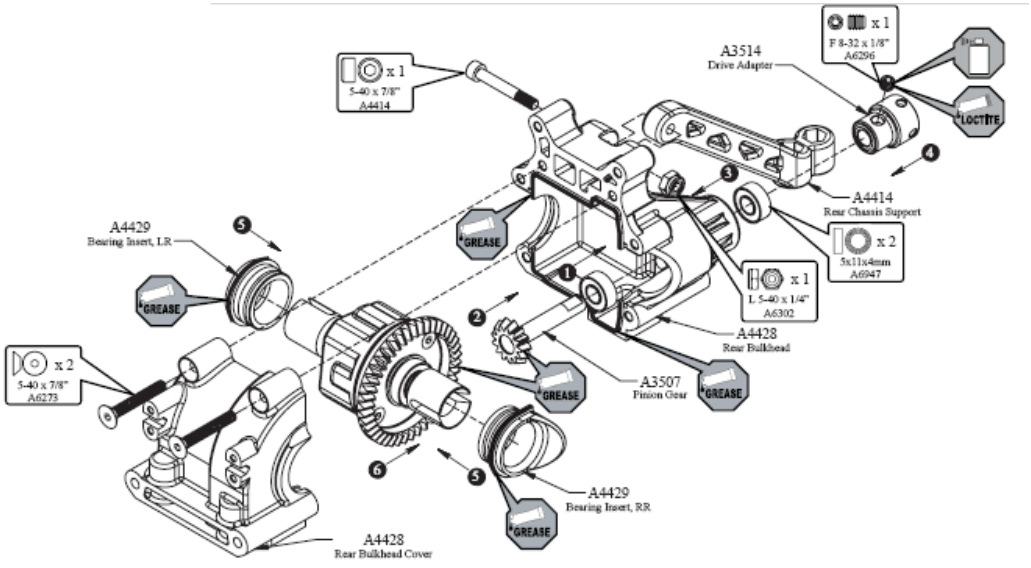
Tot objecte dintre dels assamblatges té un punt d'origen on s'aplica el sistema de referència de l'objecte.

Els graus de llibertat denoten el nombre de possibilitats cinemàtiques independents de moviment. Un cos rígid té sis graus de llibertat en termes de moviment espacial, tres d'ells són graus de llibertat translacionals (de desplaçament) i tres són graus de llibertat rotacionals (de gir).

Una de les parts fonamentals de la simulació de mecanismes són les restriccions entre peces. Les restriccions determinen la forma en què s'uneixen els components de l'assemblatge i/o limiten la llibertat de moviment respecte l'estructura o el sistema de referència de l'origen. A l'aplicar restriccions, es poden suprimir graus de llibertat o es poden limitar desplaçaments i transmissió de moviments respecte altres components.

Existeixen diferents tipus de restriccions com, per exemple, restriccions de coincidència, angulars, tangencials, d'inserció,...

Per explicar les restriccions de manera més visual es poden comparar amb els plànols de muntatge d'un disseny industrial. En aquest tipus de plànols s'especifica com s'han d'unir les peces per finalment obtenir el mecanisme o estructura desitjada (veure Figura 5.3.2).



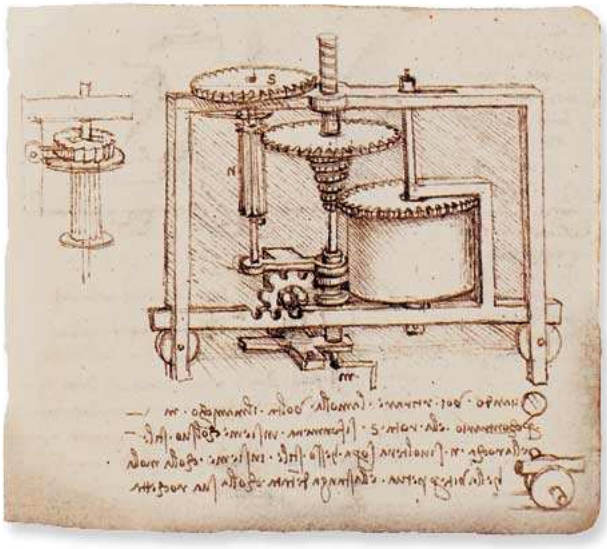
5.3.2 - Exemple de plànol de muntatge

6. Requisits del sistema

A continuació descriurem els requisits que ha de complir el sistema, tant funcionals com no funcionals. Aquest apartat ha de permetre saber tot allò que cal fer perquè el projecte compleixi tots i cadascun dels seus objectius, tant des del punt de vista de programari com de maquinari. També emmarcarem els requisits en l'entorn on es desenvolupa el projecte.

6.1 Plantejament de la problemàtica

El problema que volem solucionar és la generació de mecanismes senzills a partir d'un fitxer de configuració, generalment formats per peces com engranatges, eixos de tot tipus, motlles,... (veure Figura 6.1.1).



6.1.1 - Exemple de mecanisme simple

Un dels primers problemes consisteix en generar les peces necessàries per contruir els mecanismes de manera més àgil i automàtica.

En segon lloc, per poder simular el comportament del mecanisme, hem de connectar les peces entre elles perquè interactuïn, utilitzant restriccions. D'aquesta manera, a l'aplicar un moviment sobre el mecanisme es podrà veure el funcionament.

6.2 Plantejament de la solució

La solució que es proposa és crear un nou llenguatge amb una nomenclatura senzilla que permeti la generació de mecanismes senzills a partir de peces predefinides i la connexió entre elles a partir de les funcionalitats que proporcina la plataforma de simulació. A més, definir el comportament d'alguna de les peces com a motor de l'estructura per generar moviment.

En altres paraules, l'aplicació haurà de permetre a l'usuari carregar el codi per generar el mecanisme, o jerarquia d'assemblatges, i ser capaç de convertir el llenguatge en instruccions que la plataforma entengui per tal de generar les peces, el conjunt de sub-assemblatges, i connectar-los entre ells. A més, ha de disposar de l'opció d'activar el motor de moviment i iniciar una simulació més dinàmica.

La plataforma proporcionan totes les eines necessàries per generar les peces desde zero o per utilitzar models paramètrics que es poden modificar fins obentir el resultat desitjat. De la mateixa manera, proporciona tots els mecanismes de restriccions necessaris per tal de realitzar les connexions entre les peces, o les peces de sub-mòduls d'assemblatges

D'aquesta manera, l'usuari podrà generar un mecanisme i aplicar-li moviment sense la necessitat de conèixer la plataforma ni haver d'interactuar directament amb l'eina de simulació.

6.3 Requisits funcionals

Tot seguit es llisten totes les funcionalitats que ha de realitzar l'aplicació, és a dir, les funcionalitats a desenvolupar:

- Carregar fitxers XML
- Interpretar fitxers
    - Generació d'assemblatges
    - Generació de peces
    - Connexió de peces i/o assemblatges
    - Assignació del motor del mecanisme
    - Estructura jeràrquica d'assemblatges

6.4 Requisits no funcionals

Tot seguit es definiran les característiques de com ha de ser l'aplicació. Aquests requisits fan referència al què hem de tenir en compte a l'hora de dissenyar el sistema, com ara els recursos.

Els requisits no funcionals són:

- El llenguatge XML ha de ser eficient i fàcil d'apendre. Ha de optimitzar les operacions al màxim per fer els càlculs en el mínim de temps possible.
- El llenguatge ha de ser informatiu. Si hi ha qualsevol error s'ha d'indicar on es troba l'error per facilitar la programació.

La plataforma on s'executarà el codi en el nou llenguatge serà *Autodesk Inventor 2015*. A causa de que els models que la nostra aplicació utilitzarà per generar noves peces no s'accepten en versions anteriors, un dels requisits de l'aplicació és optimitzar l'interacció només amb la versió 2015 d'*Autodesk Inventor.* A més, els requisits de hardware seran els mínims necessaris que demana aquesta plataforma:

- Sistema Operatiu Windows 7 (SPI) de 32-bits
- 8 GB de memòria RAM
- Tipus de CPU Intel o AMD, de 64 bits

7. Estudi i decisions

En aquest capítol es descriu el maquinari i el programari utilitzats durant el desenvolupament del projecte i es justificarà la tria. Cal dir que no hem tingut la necessitat d'utilitzar llibreries addicionals per dur a terme el projecte.

7.1 Autodesk Inventor 2015

Autodesk Inventor és un paquet de modelatge paramètric de sòlids en 3D produït per l'empresa *Autodesk*. Aquest paquet es va crear per satisfer la necessitat de passar del disseny en dues dimensions a volums i alhora portar aquesta tecnologia als ordinadors personals.

Inventor està pensat per agilitzar el procés de desenvolupament d'un producte per realitzar proves mecàniques sobre un prototip virtual, reduint el temps i els costs del procés tradicional de prototipatge.

Com s'ha dit anteriorment, *Inventor* està basat en tècniques de modelat paramètric. L'usuari pot dissenyar peces i assemblar-les amb altres per crear-ne de noves. A diferència de l'*Autocad*, que només permet definir les dimensions, aquesta eina permet modelar la geometria, redefinir les dimensions i establir els materials a utilitzar, com peces metàl·liques, guardant les referències dintre el model generat. D'aquesta manera, si modifiquem les dimensions del model, automàticament s'actualitza la geometria basant-se en les noves dimensions.

Els components principals de l'*Inventor* són les peces i els assemblatges.

Les peces es creen a partir d'un esbós i de l'especificació de les seves característiques. Per exemple, per crear un cub és necessari l'esbós d'un quadrat i donar-li profunditat. Aquest disseny té l'avantatge que podem corregir les característiques i els esbossos sempre que volguem. Per contra, en els models de versions d'*Autodesk Inventor* anteriors s'havia de començar de zero cada vegada que es volien corretgir característiques.

Els assemblatges consisteixen en peces o altres assemblatges carregats en un mateix espai i connectats, o no, entre si. Per connectar les peces s'afageixen restriccions als models, superfícies, arestes, plans, punts o eixos. Podem distingir els següents tipus de restriccions:

- Coincidència
- Nivelació
- Inserció
- Angle
- Tangent
- Transicional
- Moviment de rotació i/o translació
- Sistema de coordenades de l'usuari

Aquest sistema de modelat permet crear assemblatges de grans dimensions gràcies a que el sistema de peces pot estar generat per altres assemblatges més petits.

Inventor utilitza formats específics d'arxius per les peces (.ipt), assamblatges (.iam), vista del dibuix (.idw i .dwg) i presentacions (.ipn), però el format de l'arxiu d'AutoCAD .dwg pot ser importat/exportat com un esbós.

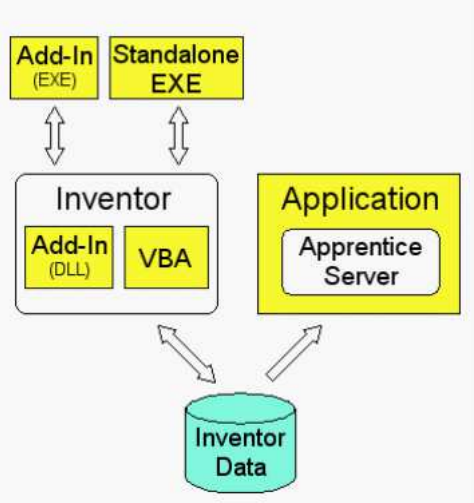
Inventor permet a l'usuari interactuar sobre l'editor (veure figura 7.1.1) per crear els mecanismes. Al mateix temps ofereix funcionalitats per a usuaris amb coneixements informàtics per escriure programes que interactuaran amb la plataforma, ampliant les possibilitats que ofereix l'editor visual.

7.1.1 - Entorn de treball Autodesk Inventor

L'API d'*Autodesk Inventor* està creada utilitzant la tecnologia de Microsoft anomenada "*Automation*". Aquesta tecnologia es fa servir en molts altres programes de Microsoft, com Word, Excel o AutoCAD. Un dels principals avantatges d'aquesta API és que permet programar-hi en la majoria de llenguatges de programació actuals, com Visual

Basic, Visual C++, Visual C#, Delphi, Perl, Java, entre altres. A més a més, les funcionalitats s'exposen de la mateixa manera que la programació orientada a objectes, molt més fàcis d'utilitzar que les APIs orientades a funcions.

Existeixen diverses formes d'accedir a l'API per donar pas al programador a decidir quina és millor per al programa a desenvolupar. Les maneres d'accedir-hi són via VBA, Add-Ins, EXE independent o Apprenctice Server.



7.1.2 - Il·lustració de les vies d'accés a l'API

VBA, o *Visual Basic for Aplications*, és un entorn de programació que s'accedeix des de l'interior d'*Autodesk Inventor*. Els programes que utilitzan VBA s'hi refereixen amb freqüència com a "macros". VBA és utilitzat habitualment pels usuaris finals per escriure petits programes per automatitzar tasques repetitives, encara que certament no es limita a això. Un programa de VBA té el mateix accés a totes les funcions de l'API com qualsevol dels altres mètodes d'accés a l'API.

Utilitzar VBA per accedir a l'API disposa de certs avantatges. En primer lloc, VBA s'entrega amb *Autodesk Inventor*. En segon lloc, és capaç d'integrar els programes dins dels documents d'*Autodesk Inventor*. Si tenim un programa que utilitza informació específica, aquesta via disposa de la manera per mantenir el programa amb les dades de com han estat dissenyat per al seu ús. En tercer lloc, VBA s'executa en el mateix procés que *Autodesk Inventor*, per tant, s'obtenen els avantatges d'estar comunicats de forma més estreta.

Add-Ins són un tipus especial de programa que és capaç de fer coses que altres mètodes d'accés a l'API no poden fer. En primer lloc, *Autodesk Inventor* inicia el complement de forma automàtica. En segon lloc, un Add-In és capaç de crear comandes. Amb aquestes dues característiques té el mateix accés per utilitzar l'API com els altres programes escrits que utilitzen altres mètodes d'accès.

El fet que els complements es puguin iniciar automàticament cada vegada que s'executa *Autodesk Inventor* és una capacitat molt important per a molts programes. La majoria de les aplicacions que volen oferir una perfecta integració amb *Autodesk Inventor* s'implementen com a Add-In degut a que, d'aquesta manera, la funcionalitat estarà disponible per a l'usuari cada vegada que es posi en marxa l'entorn.

A la figura 7.1.2 podem veure que el mètode d'accés Add-In apareix dues vegades: com un arxiu DLL i com un EXE. Amb un Add-In, es pot crear un arxiu DLL, que es

desenvoluparà en el mateix procés que *Autodesk Inventor*, o un arxiu EXE, que es desenvoluparà en un procés separat. La majoria d'Add-Ins s'escriure com DLL per aprofitar un major rendiment. Per altra banda, un EXE és sobretot beneficiós per a la depuració.

Els Add-Ins es poden escriure amb qualsevol llenguatge que suporti la creació d'arxius ActiveX EXE o DLL, com Visual C++ i Visual Basic. Els Add-Ins no es poden crear amb VBA.

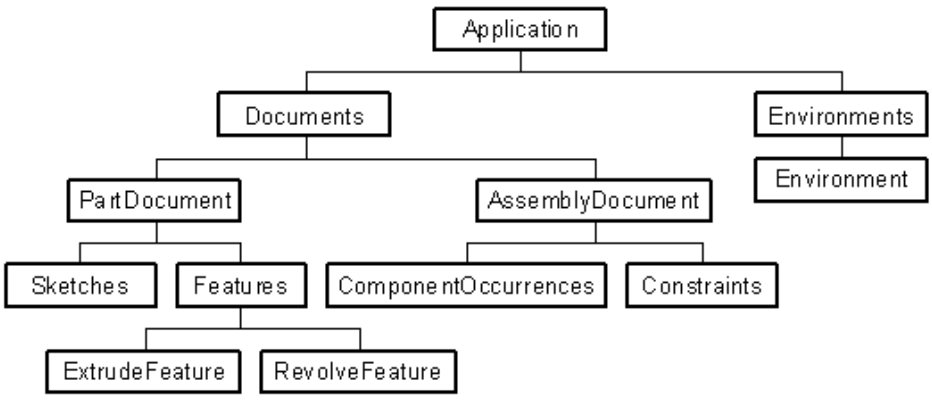
Un **EXE independent** és un programa extern que es connecta a *Autodesk Inventor*. Aquest tipus de programes acostumen a utilitzar-se quan el programa utilitza *Autodesk Inventor*, però sense utilitzar la seva interfície perquè ja disposen d'una pròpia i no és necessari que l'usuari interactui amb l'eina.

L'EXE independent s'executa fora del procés d'*Autodesk Inventor*, fet que causa una mínima pèrdua de rendiment. Per aquest motiu només interessa per a processos poc interactius.

L'***Apprentice*** és un servidor ActiveX que es pot utilitzar dins d'altres aplicacions per proporcionar accés a les dades d'*Autodesk Inventor*. *Apprentice* és essencialment un subconjunt d'*Inventor* que s'executa amb altres aplicacions. A més, no disposa d'una interfície d'usuari. L'única manera d'interectuar amb l'aprenent és a través de l'API. Aquesta via proporciona accés a l'estructura del conjunt, B-Rep, la geometria i les propietats del fitxer. A més, l'accés a la informació és de només lectura.

L'*Apprentice* és útil en qualsevol aplicació independent que necessiti accés a la informació continguda en els documents d'*Autodesk Inventor*. És molt més eficient que utilitzar l'*Inventor* per realitzar les mateixes operacions gràcies a que no té una interfície, cosa que li permet realitzar operacions molt més ràpid.

Com s'ha dit anteriorment, l'API d'*Autodesk Inventor* està orientada a objectes. Això vol dir que disposem d'un conjunt d'objectes els quals contenen mètodes i propietats



7.1.3 - Jerarquia simplificada d'objectes

per realitzar consultes o modificar-los. Per tant, a continuació es mostrarà la jerarquia d'objectes simplificada que formen l'API.

Com podem veure a la figura 7.1.3, l'objecte principla és l'*Application*, que disposa del control de la plataforma. A partir d'aquest objecte es pot accedir als *Documents*, que són els objectes principals on podem treballar, i als entorns *Environments.* Distingim dos tipus principals de documents: els *PartDocument*, que consisteix en un document on podem definr peces individuals, i els *AssemblyDocument*, que serveix per assamblar peces entre si. Aquesta estructura segueix el patró composite.

Finalment, disposem d'altres objectes per poder treballar amb els documents: els *Sketches*, o esbossos, que permeten definir les mesures de la peça en el pla, i els *Features*, que permeten donar volum a les peces. Dintre els *AssemblyDocument* podem treballar amb els *Contraints*, que defineixen principalment les unions entre peces, entre altres.

Motius de selecció

Hem decidit fer servi aquesta IDE en lloc d'una altra per diferents motius. En primer lloc, estem parlant d'una IDE molt potent que disposa de moltes funcionalitat i ajudes dintre el mateix entorn. A més, disposa d'eines i mètodes d'integració, com s'ha explicat anteriorment, que permet poder treball interna i externament amb altres programes. Per a obtenir informació, disposa d'una comunitat molt gran online, la qual cosa facilita l'obtenció de respostes i un punt per preguntar dubtes. Per altra banda, el fet de ser una eina gratuïta per a estudiants i docents fa que sigui molt temptador poder utilitzar-la.

El fet d'escullir la versió 2015 és, a part de ser una de les últimes versions llençades al mercat, és per motius de compatibilitat amb el computador destinat a realitzar el projecte.

Per tant, degut a la seva complexitat i suport, s'ha decidit utilitzar *Autodesk Inventor 2015.*

7.2 Python vs C#

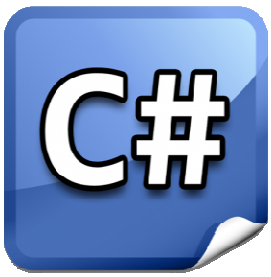
1ra opció: Python

Python és un potent llenguatge de programació fàcil d'apendre i àmpliament utilitzat per a propòsits generals de tot tipus. Compta amb estructures de dades d'alt nivell eficients i en un enfocament simple, però eficaç, que suporta diversos paradigmes de

programació, incloent-hi programació orientada a objectes, imperativa i també funcional o procedimental. La seva filosofia de disseny busca llegibilitat en el codi, i la seva sintaxi permet als programadors expressar conceptes en menys línies de codi del que seria possible en llenguatges com C++ o Java. Gràcies a la sintaxi elegant i el tecleig dinàmic, juntament amb la seva naturalesa interpretada, fan que Python sigui un lleguatge ideal per generar seqüències ordenades (scripting) i desenvolupar de manera ràpida aplicacions en moltes àrees, i en la majoria de plataformes. També proveeix estructures per permetre programes més entenedors, tan en petita com a gran escala.

Python disposa d'un interpret i una extensa quantitat de llibreries estàndard disponibles de forma gratuïta per a multiplataforma, i poden ser distribuïts lliurement. Es poden trobar llibreries par a càlculs matemàtic o científics, per generar entors gràfic o videojocs, o per conexions entre aplicacions utilitzant sistemes COM, entre molts d'altres. A més, existeix moduls de tercers, programes, eines i documentació addicional per ampliar el coneixement i l'abast del llenguatge Python.

2na opció: C#



C# és un llenguatge de programació orientada a objectes desenvolupat i estandaritzat per Microsoft com a part de la plataforma .NET, que més endavant va ser aprovat com un estàndard per ECMA i ISO. C# és un dels llenguatges de programació dissenyat per la Infraestructura de llenguatge comú (*Common Language Infrastructure* o CLI).

La sintaxis del C# deriva del C/C++ i utilitza el model d'objectes de la plataforma .NET, similar al de Java, tot i que inclou millores d'altres llenguatges. La intenció de C# és ser un llenguatge simple, eficaç, amb seguretat de tipus, de proposit general i orientat a objectes. Les nombroses innovacions de C# permeten desenvolupar aplicacions ràpides i mantenir la expresivitat i elegància dels llenguatges d'estil de C.

Motius de selecció

Tot i que la majoria d'exemples i models de programes que interactuen amb la plataforma d'*Autodesk Inventor* utilitzen VB o C# com a llenguatges de programació, vam decidir escollir Python com a llenguatge de programació del projecte per motius molt variats. Per una banda, Python és un llenguatge modern i molt complet, alhora de ser molt senzill d'utilitzar i apendre, que permet desenvolupar quasi qualsevol codi. A

més, disposa d'una nombrosa quantitat de llibreries que realitzen tasques ja desenvolupades per tercers. Cal dir que també hi havia la motivació d'escullir-lo per poder apendre'l i ampliar coneixements utilitzant un nou llenguatge de programació.

Cal dir que, degut a que la utilització de Python per accedir a Inventor requeria utilitzar una llibreria COM externa no oficial amb exemples molt simples, vam considerar oportú, i encertat, establir un segon llenguatge més fortament testejat per a la customització d'*Autodesk Inventor* amb el qual poguessim desenvolupar el projecte en cas de que el primer llenguatge, o les llibreries, fallessin.

Per aquesta segona opció, es va escollir el llenguatge C#, d'entre els més utilitzats per customitzar *Autodesk Inventor,* per les seves característiques, com ser un llenguatge orientat a objectes, i per ja disposar de coneixements previs tan de C com de C++.

7.3 Llibreria Win32com

Aquesta llibreria s'encarrega de la connexió entre el programa en llenguatge Python amb un altre d'extern, basat en la tecnologia COM (Component Object Model), una plataforma de Microsoft utilitzada per permetre la comunicació entre processos i la creació dinàmica d'objectes, en qualsevol lleguatge que la soporti.

L'objectiu del disseny de la llibreria és fer que el programador no tingui la necessitat de saber més intensament com funciona la llibreria amb la qual es vol connectar. Simplement es demana un identificador IID, una versió i un LCID per identificar el tipus de la llibreria, en aquest cas, els identificadors i versions d'*Autodesk Inventor*. A partir d'aquesta informació, obtindrem un modul de Python per a dur a terme la connexió entre els programes i saber els mètodes i funcions de la llibreria remota. El modul de win32com.client.gencache implementa totes aquestes funcionalitats.

7.4 Visual Studio 2013

Microsoft Visual Studio és un entorn de desenvolupament integrat (IDE, per les sigles en anglès) de *Microsoft.* S'utilitza per desenvolupar programes d'ordinador per la família de sistemes operatius *Microsoft Windows,* tot i que inclou la opció d'integrar altres llenguatges, com Pyton. També permet desenvolupar pàgines web, aplicacions web i serveis web.

Visual Studio inclou un editor de codi amb suport de *IntelliSense* i *code refactoring*. El *debugger* que porta incorporat permet treballar tant a alt nivell com a nivell de màquina. Altres eines integrades que incorpora són: formularis de disseny per aplicacions amb interfície gràfica, dissenyador web, dissenyador de classes i un dissenyador d'esquemes de bases de dades. *Visual Studio* accepta *pluig-ins* per incorporar funcionalitats a quasi tots els nivells. Per exemple, permet incorporar eines per control de versions (com *Subversion* o *Visual SoruceSafe*), per treballar amb nous llenguatges, per gestionar el cicle de desenvolupament del softwar,... .

Visual Studio suporta diferents llenguatges de programació i permet editors de codi i suport per *debugger* de quasi qualsevol llenguatge de programació. Els llenguatges incorporats per defecte són C, C++ i C++/CLI (via Visual C++), VB.NET (via Visual Basic .NET), C# (via Visual C#), i F# (des de Visual Stuido 2010). També inclou la opció d'incorporar altres llenguatges com M, Python, Ruby i CUDA, XML/XSLT, HTML/XHTML, JavaScript i CSS.

Microsoft proporciona versions "Express" de *Visual Studio* sense cap cost. També proprociona de forma gratuïta llicències comercials de *Visual Studio*.

Motius de selecció

Hem decidit fer servir aquesta IDE, i no una altra, per diverses raons. Primerament, estem parlant d'una IDE amb molts anys de desenvolupament i, per tant, amb moltes funcionalitats. També té una comunitat molt gran darrera, la qual cosa ens permet trobar solucions a problemes amb la IDE de forma molt més ràpida. Per altra banda, el fet de ser gratuïta ens ha estat molt temptador.

Per tant, al ser una eina molt completa i amb el suport intert de Microsoft i de lal comunitat d'internet, l'entorn de desenvolupament escollit ha estat *Microsoft Visual Studio 2013.*

7.5 Sistema Operatiu

Per desenvolupar el projecte s'ha utilitzar una de les últimes versios del sistema operatiu de Windows, concretament la versió 8.1.

Motius de selecció

Els motius de selecció que ens han portat a escollir Windows 8.1 com a sistema operatiu per desenvolupar el projecte han estat, pràcticament, a causa de ser el

sistema predefinit de la nostra màquina. A més, és un dels sistemes operatius més estés i amb completa compatibilitat amb totes les eines i programes que s'han utilitzat.

7.6 GANTT Project

Gantt Project és una aplicació de programari lliure que permet realitzar diagrames de Gantt. Un diagrama de Gantt és una eina gràfica que té com a objectiu veure fàcilment la planificació d'un projecte mostrant el temps de dedicació per cadasccuna de les tasques.

El diagrama de Gantt no indica les relacions exixtent entre activitats, però la posició de cada tasca al llarg del temps fa que es puguin identificar aquestes relacions i dependències.

Motius de selecció

Hem escollit aquest programa perquè és de programari lliure i els resultats que s'obtenen són de qualitat.

7.7 StarUML

StarUML és un projecte open source per crear ràpidament diagrames UML. Aquesta aplicació està en continua expansió degut a la seva filosofia de distribució i és totalment gratuïta.

Motius de selecció

El principal motiu per escullir aquesta aplicació, a part de que ens permet desenvolupar els diferents diagrames necessaris per el projecte, ha estat el fet de ser gratuïta i no necessitar d'una llicència per la seva utilització, com moltes altres aplicación similars.

8. Anàlisis i disseny

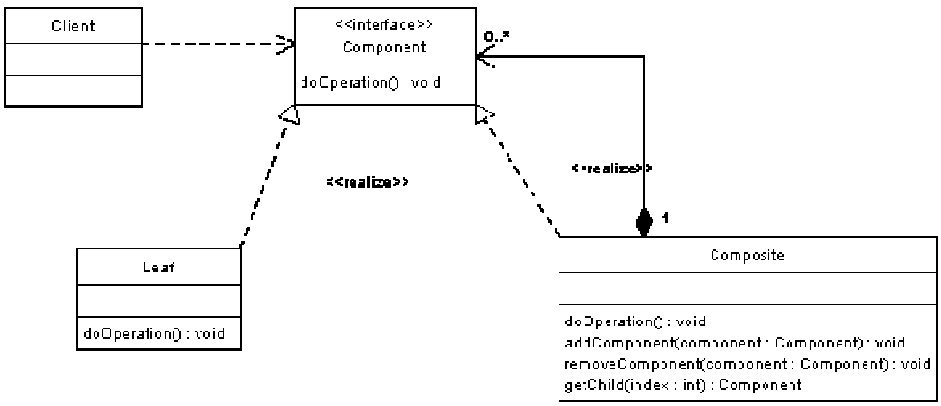
A continuació veurem de manera detallada les necessitats del sistema i les solucions desenvolupades per a cada una de les necessitats. Es realitzarà l'anàlisis i el disseny del sistema a partir de diferents diagrams i esquemes, com model de dades i de processos.

L'objectiu del projecte és poder desenvolupar un sistema que permeti generar, a partir d'un fitxer, mecanismes senzills a la plataforma escullida, l'*Autodesk Inventor*, simular-los i poder treballar lliurament amb ells. Per aquest motiu, es desenvoluparà un llenguatge en fitxers xml que permeti carregar i posicionar una jerarquia d'assemblatges i peces. En altres paraules, l'aplicació permetrà crear noves peces i connectar-les entre elles especificat la posició i poder aplicar moviment sobre l'estructura per simular el seu funcionament.

8.1 Introducció al llenguatge XML

Per a la generació dels assamblatges, necessitarem un fitxer XML, fàcil d'utilitzar i redactar, que contindrà les estructures o peces a carregar, el posicionament entre elles i, si es volgués, l'assignació d'un motor.

Aquests fitxers seguiran un patró de disseny anomenat **Composite**, encarregat de compondre objectes en estructures d'arbre per representar una jerarquia parcial-total. En altres paraules, aquest patró permet tractar objectes i conjunts d'objectes individuals de manera uniforme.



8.1.1 - Patró de disseny Composite

Els fitxers d'aquest projecte podran contenir només peces, només referències a altres fitxers o una combinació de les dues. L'estructurà sintàctica, que contindrà la capçalera per la comanda *assembly,* serà la següent:

- **Load**: El primer punt a trobar, de manera opcional, per carregar altres fitxers XML. Els paràmeteres a definir són:

  - **name_as**: L'assamblatge que es carrega, dintre la gerarquia, rebrà el nom indicat en aquest paràmetre.
  - **dir_assembly**: Aquest paràmetre es subdivideix en dos paràmetres més:
    - **dir**: nom del fitxer a carregar
    - **use_motor**: Booleà que indica si es vol utilitzar o no el motor de l'assamblatge que s'està carregant, si en tingués.
- **Creation**: En aquest apartat s'indiquen els paràmetres necessaris per construir una peça. Les variables són:
  - **name:** Aquest paràmetre indica el nom que rebrà l'element dintre l'assamblatge. Poden haver altres peces amb el mateix nom sempre que estiguin en altres assamblatges.
  - **part_info**: Aquest paràmetre es subdivideix en dos tipus de paràmetres més:
    - **part**: Indicació del tipus a construir, que poden ser: new gear, new axis, new bevel.
    - **param**: segons el tipus a construir, hi haurà un o més paràmetres "param" dintre les **part_info**. La construcció d'un *gear* requereix indicar el diametre intern de la peça, la construcció d'un *axis* requereix indicar l'amplada i la llargada d'aquest, i la construcció d'un *bevel* requereix indicar el número de dents.
- **Operation**: En aquesta especificació s'indica el tipus d'operació que es vol realitzar respecta les peces indicades. Els tipus de paràmetres necessàris són:
  - **constraint**: S'indica el tipus d'acció a realitzar. Actualment només hi ha l'opció d'unió (join).
  - **param**: S'han d'especificar tres variables "param", amb els següents valors:
    - El primer paràmetre indica el nom de la primera peça a utilitzar.
    - El segon paràmetre indica el nom de la segona peça a utilitzar.
    - El tercer paràmetre indica, en cas que un dels elements anteriors és un eix, la distància respecta la base de l'eix on s'uneix l'engranatge. Si les dues peces són engranatges del mateix tipus, s'indica l'angle, en graus, que la segona peça es col·locarà respecta la primera peça.
- **Motor**: Per acabar, sindica, si es volgués, la peça que tindrà la funció d'ancla i motor de dintre l'assamblatge. Només es necessita un únic paràmetre:
  - **motor_name**: Indica el nom de la peça a utilitzar.

Cal dir que els noms de referència que es volen utilitzar, però es troben dintre altres assamblatges, s'indica l'adreça concatenant amb un punt el nom dels assamblatges i acabant amb el nom de la peça desitjada.

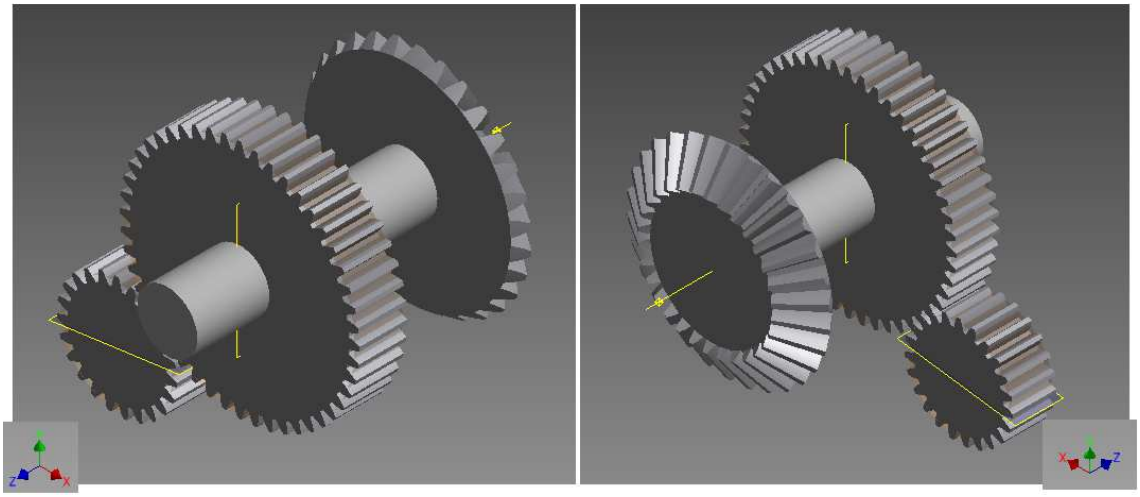
A continuació veurem un exemple de dos fitxers. El primer carrega dos engranatges simples i un eix. El segon utilitza l'assemblatge anterior per connectar un *bevel* amb l'eix ja creat. Primer de tot s'especificarà la capçalera del XML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<assembly xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
                                        xsi:noNamespaceSchemaLocation="assembly.xsd">
  //Bloc de creació per l'engranatge 1 (diàmetre 10 cm)
  <creation>
    <name>g1</name>
    <part_info>
      <part>new gear</part>
      <param>10</param>
    </part_info>
  </creation>
  //Bloc de creació per l'eix 1 (diàmetre 2.5 cm, longitud 10 cm)
  <creation>
    <name>a1</name>
    <part_info>
      <part>new axis</part>
      <param>2.5</param>
      <param>10</param>
    </part_info>
  </creation>
  //Bloc de creació per l'engranatge 2 (diàmetre 5 cm)
  <creation>
    <name>g2</name>
    <part_info>
      <part>new gear</part>
      <param>5</param>
    </part_info>
  </creation>
  //Bloc d'unió entre l'engranatge 1 i l'eix 2 a 5 cm
  <operation>
    <constraint>join</constraint>
    <param>g1</param>
    <param>a1</param>
    <param>5</param>
  </operation>
  //Bloc d'unió entre l'engranatge 1 i el 2 a 136 graus
  <operation>
    <constraint>join</constraint>
    <param>g1</param>
    <param>g2</param>
    <param>136</param>
  </operation>
  //Engranatge 2 amb funció de motor
  <motor>
    <motor_name>g2</motor_name>
```

```
    </motor>
</assembly>
```

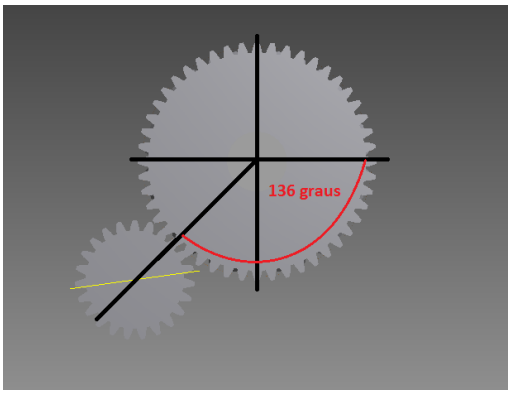
```
<?xml version="1.0" encoding="ISO-8859-1"?>

<assembly xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                        xsi:noNamespaceSchemaLocation="assembly.xsd">
    //Carregar assemblatge 1 del primerExemple.xml (sense motor)
    <load>
        <name_as>as1</name_as>
        <dir_assembly>
            <dir>primerExemple.xml</dir>
            <use_motor>false</use_motor>
        </dir_assembly>
    </load>
    //Crear engranatge cònic 1
    <creation>
        <name>b1</name>
        <part_info>
            <part>new bevel</part>
            <param>30</param>
        </part_info>
    </creation>
    //Unir engranatge cònic 1 amb eix 1 de l'assemblatge 1 a 10 cm
    <operation>
        <constraint>join</constraint>
        <param>b1</param>
        <param>as1.a1</param>
        <param>10</param>
    </operation>
    //Engranatge 1 de l'assemblatge 1 amb funció de motor
    <motor>
        <motor_name>as1.g1</motor_name>
    </motor>
</assembly>
```

El resultat esperat per aquest exemple serà la unió d'un engrenatge as1.g1 amb un engrenatge as1.g2 en una posició de 136 graus respecte el primer. A més, el primer engranatge as1.g1 estarà situat just al centre d'un eix a1 que, alhora, tindrà un engranatge cònic a un dels extrems. Podem veure la imatge del resultat a les figures 8.1.2 i 8.1.3.



8.1.2 - Visió frontal i inversa del resultat

Pel què fa el posicionament d'un engranatge respecte un altre a partir d'un valor en graus, es fa seguint el sentit de les agulles del rellotge i començant des del valor positiu de l'eix de les X de l'engranatge de referència (veure Figura 8.1.3).



8.1.3 - Exemple de posicionament per graus

Podem veure que l'engranatge g1 (el de diàmetre 10) de l'exemple serà el de referència, i el g2 (el de diàmetre 5) serà el que es posicionarà al seu voltant en un angle de 136 graus respecte l'eix de les X i en sentit horari.

8.2 Fitxes i diagrames de cas d'ús

Un cas d'ús és una tècnica per a la captura de requisits potencials d'un nou sistema. Cada cas d'ús proporciona un o més escenaris que indiquen com hauria d'interactuar el sistema amb l'usuari, o amb un altre sistema, per aconseguir l'objectiu desitjat.

Com que tractem amb fitxers XML molt fàcils d'entendre i implementar, els actors que interactuaran amb el nostre sistema seran qualsevol tipus d'usuari (veure figura 8.2).

8.2.1 - Actor del sistema

8.2.1 Diagrama de cas d'ús general

El diagrama de cas d'ús següent (figura 8.2.2) es mostren els requisits principals del sistema.

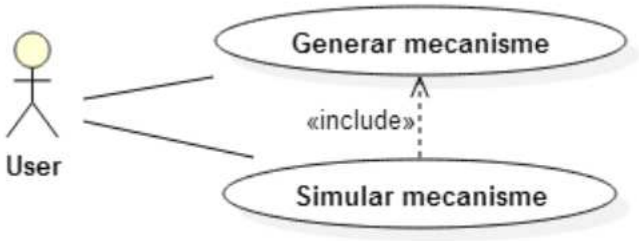


Figura 8.2.2 - Diagrama de cas d'ús general

A continuació es mostren les fitxes de cada cas d'ús del sistema. Aquestes ens permetran tenir una idea del funcionament general de cada cas d'ús.

Fitxa de cas d'ús	**Generar mecanisme**
Descripció	El dissenyador té un codi i vol executar-lo
Actor	Usuari
Precondició	Té un arxiu XML amb el codi programat amb o sense errors
Flux principal	1- Clicar el botó d'executar arxiu 2- Obrir el fitxer amb el codi 3- Executar el codi a través del programa
Flux alternatiu	3- Hi han errors en el fitxer XML 3.1- Corregir l'error 3.1- Tornar a generar mecanisme
Postcondició	Les peces apareixen unides a la plataforma *Autodesk Inventor*

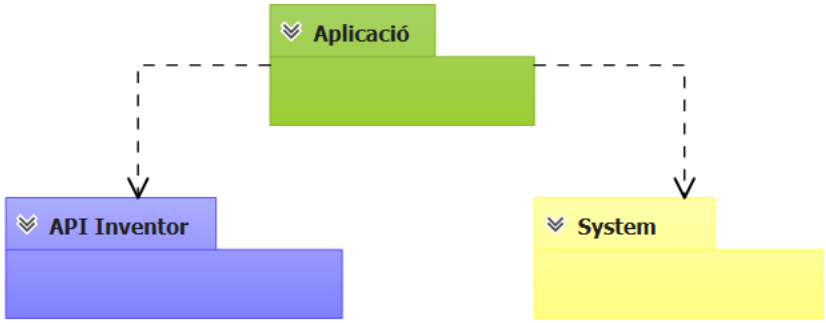
Fitxa de cas d'ús	**Simular mecanisme**
Descripció	El dissenyador vol comprovar el funcionament del mecanisme simulant el moviment
Actor	Usuari
Precondició	S'ha generat prèviament un mecanisme amb motor
Flux principal	1- Clicar el botó de simulació
Flux alternatiu	
Postcondició	Es mostren les peces movent-se a la plataforma *Autodesk Inventor* impulsades per la peça motor

8.3 Diagrama de classes

Un diagrama de classes és un tipus de diagrama d'estructura estàtica que descriu l'estructura d'un sistema mostrant els atributs, les classes i les relacions entre elles. En aquest apart es definirà el contingut i les funcions de les classes desenvolupades pel projecte i només el contingut i les funcions de les classes utilitzades d'altres llibreries, com són l'API d'*Autodesk Inventor* i les pròpies del llenguatge C#.

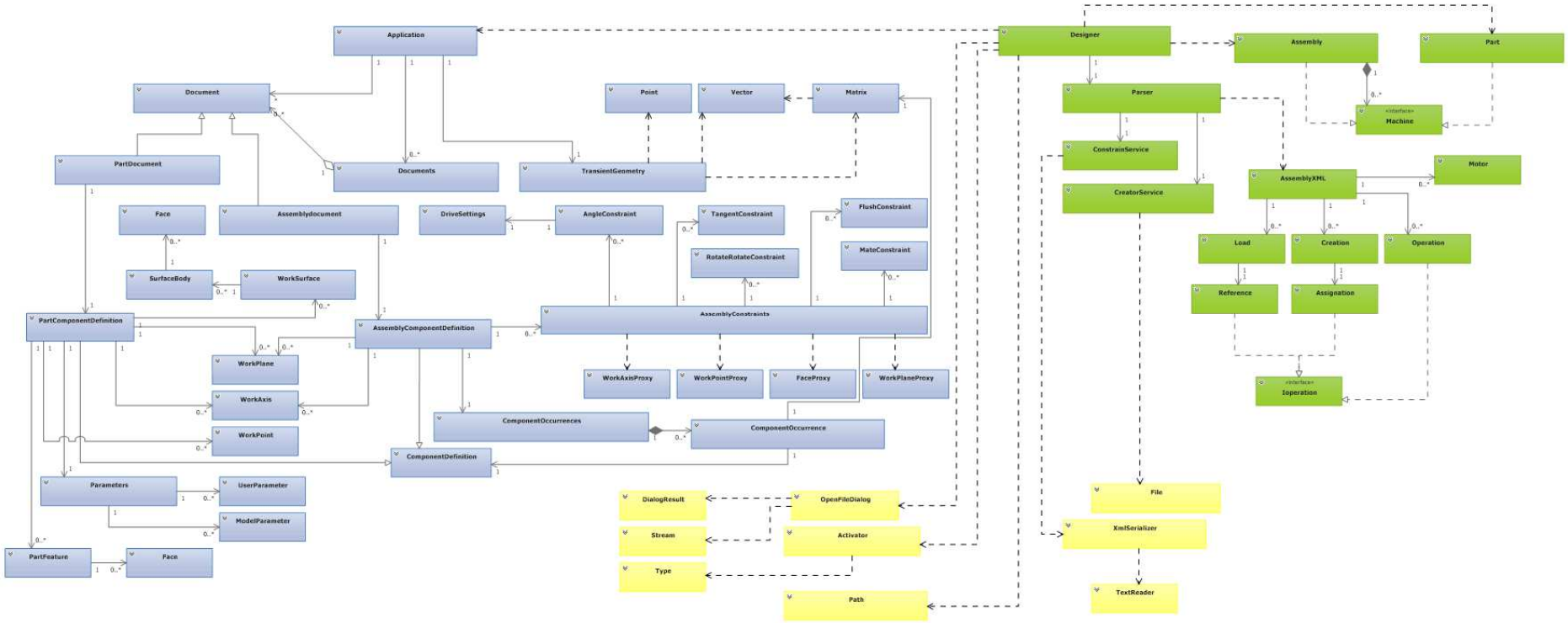
8.3.1 Mòduls funcionals

L'aplicació es divideix en tres mòduls principals, com es mostra a la figura 8.3.1. Aquests mòduls són les diverses llibreries pròpies de C# (llibreries del paquet *System*), l'API d'Inventor i l'aplicació pel projecte desenvolupat.



8.3.1 - Mòduls del projecte

A continuació es pot veure el contingut d'aquests tres mòduls de manera general, distingint per color el mòdul d'on pertany cada classe (figura 8.3.2).



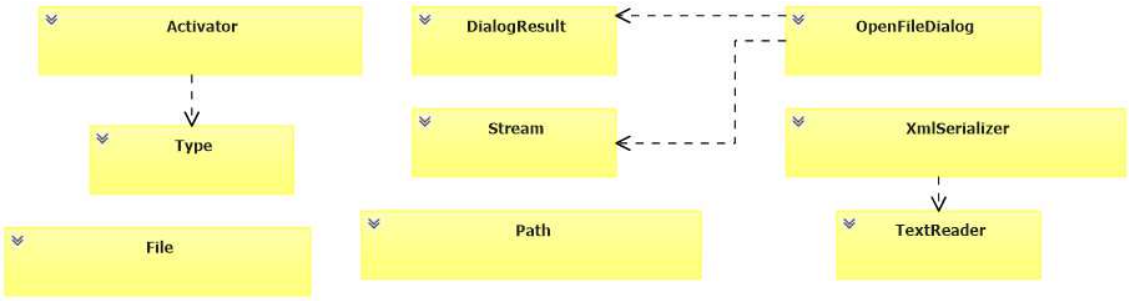
8.3.2 - Diagrama de classes reduit

A causa de la complexitat i el nombre de classes d'altres fonts utilitzades en aquest projecte, només es farà referència a les variables i mètodes emprats de cada una d'elles, encara que la classe original de l'API proporsioni més paràmetres o funcions. En els següents apartats s'explicarà cada un dels mòduls per separat.

8.3.2 Mòdul de System

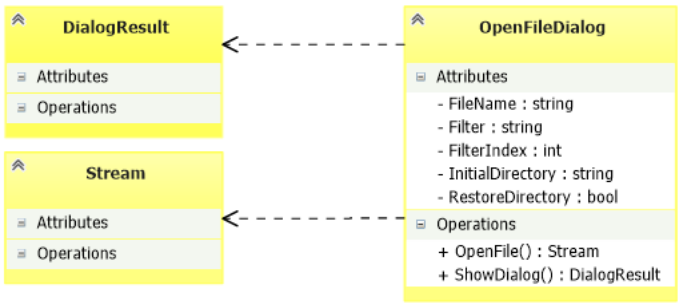
L'espai de noms de System (*namespace System*), que per defecte forma part del llenguatge C#, conté les classes funamentals i les classes base que defineixen l'ús comú dels valors i referències als tipus de dades, events i capturador d'esdeveniments, interfícies, atributs i processador d'excepcions. Altres classes del mòdul proveeixen serveis amb suport per convertir tipus de dades, manipular paràmetres dels mètodes, matemàtiques, invocació de programes en local i remot, manipulació de l'entorn de l'aplicació i supervisió d'aplicacions manipulades o sense manipular.

De tot el conjunt de classes que comporta l'espai de noms de *System*, només s'han utilitzat les classes que es poden veure a la figura 8.3.4.



8.3.4 - Conjunt de classes del mòdul System

OpenFileDialog

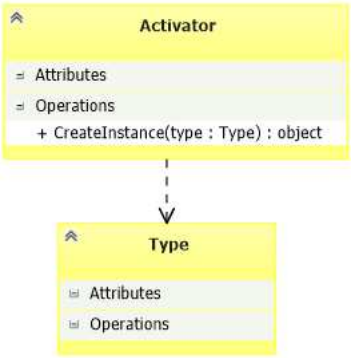


La crida a aquesta classe permet a l'usuari seleccionar, a través d'una finestra emergent, un fitxer per obrir-lo.

- *ShowDialog() : DialogResult* - És l'encarregat d'obrir la finestra de selecció, que utilitzarà els atributs de la classe, com Filter, per mostrar només certs arxius, o InitialDirectory, per obrir el directori indicat.

- *OpenFile() : Stream* - Un cop seleccionat el fitxer a obrir, aquesta comanda genera un *Stream* amb tota la informació del fitxer.

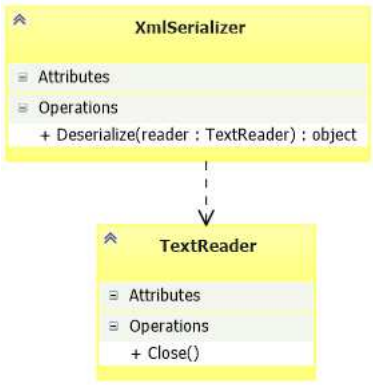
Activator



Aquesta classe s'encarrega de generar la instància i connexió amb l'aplicació desitjada, en aquest cas amb l'*Autodesk Inventor*.

- *CreateInstance(type : Type) : object* - Donada una classe Type, generada a partir de l'identificador d'*Autodesk Inventor*, es crea la referència de l'aplicació d'Inventor, accés principal per accedir a l'API.

XmlSerializer



Aquesta classe permet serialitzar o desserialitzar objectes a i desde documents XML, permetent controlar com es realitzarà aquest procés.

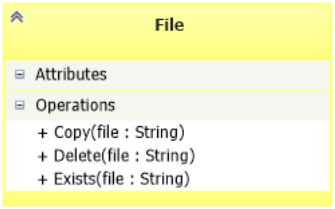
- *Deserialize(reader : TextReader) : object* - Aquest mètode espera rebre un *TextReader,* que correspondrà a un document XML, per obtenir un objecte equivalent al contingut XML (en aquest cas, donarà com a resulta un objecte *AssemblyXML*). Un exemple simple d'equivalència entre les classes esperades i la forma del document XML és el següent:

```
[Serializable]
[XmlRoot("MyClass")]
public class LaMevaClasse{
    [XmlElement("MyObjectProperty")]
    public ElMeuObjecte propietats;
}
[Serializable]
[XmlRoot("MyObjectProperty")]
public class ElMeuObjecte {
    [XmlElement("ObjectName")]
    public string NomObjecte;
}
```

```
<MyClass>
    <MyObjectProperty>
        <ObjectName>This is a object property name example</ObjectName>
    </ MyObjectProperty>
</MyClass>
```

Com podem veure, qui determina l'equivalència entre el fitxer XML i l'objecte que es vol crear (o d'on es parteix), són les annotacions de sobre les classes i els mètodes. D'aquesta manera, la funció Deserialize() és capaç de realitzar tot el procés i convertir un fitxer XML en objectes útils per utilitzar dintre l'aplicació, tinguent com a referència el nom indicat (entre altres propietats) de les annotacions.

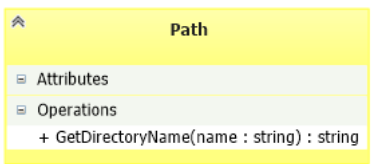
File



Aquesta classe proporciona mètodes estàtics per crear, copiar, eliminar, moure i obrir un fitxer individual.

- *Copy(file : String)* - copia el fitxer amb l'adreça *file.*
- *Delete(file : String)* - elimina el fitxer amb l'adreça *file*.
- *Exists(file : String) : bool* - retorna cert si existeix un fitxer amb l'adreça *file*.

Path



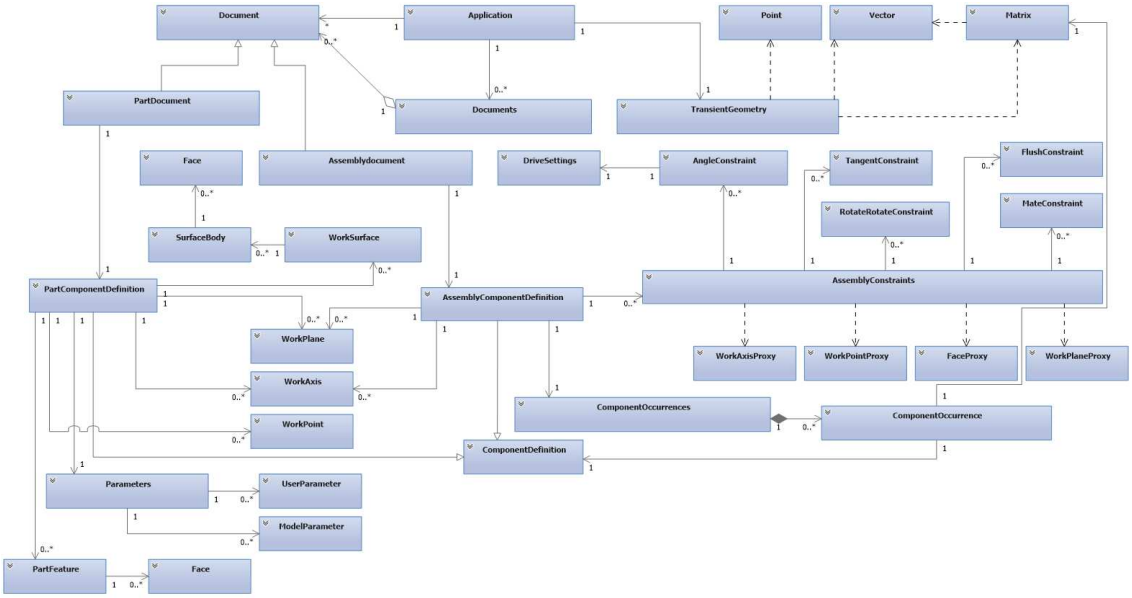
Representes operacions d'instàncies d'String que contenen informació d'adreces d'arxius o directoris.

- *GetDirectoryName(name : string) : string* - mostra el directory, sense el nom de l'arxiu, indicat per *name*.

8.3.3 Mòdul d'Inventor

Amb el mòdul d'Inventor ens referim a l'API d'*Autodesk Inventor* proporcionada per la mateixa plataforma. Aquesta API és molt extensa i, tot i ser de codi tancat, hi ha moltes ajudes per poder coneixer i utilitzar les classes i mètodes que la formen. Amb aquesta API podem gestionar, modificar, crear,... tots els elements de la plataforma on simularem els mecanismes.

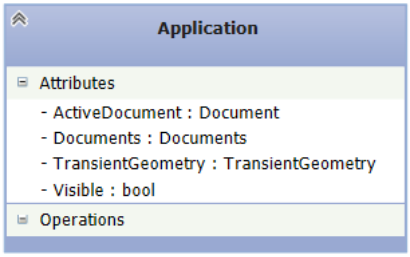
De tot el conjunt que comporta l'API d'*Autodesk Inventor*, només s'han utilitzat les classes, mètodes i variables que es poden veure a la figura 8.3.5.



8.3.5 - Mòdul d'Autodesk Inventor

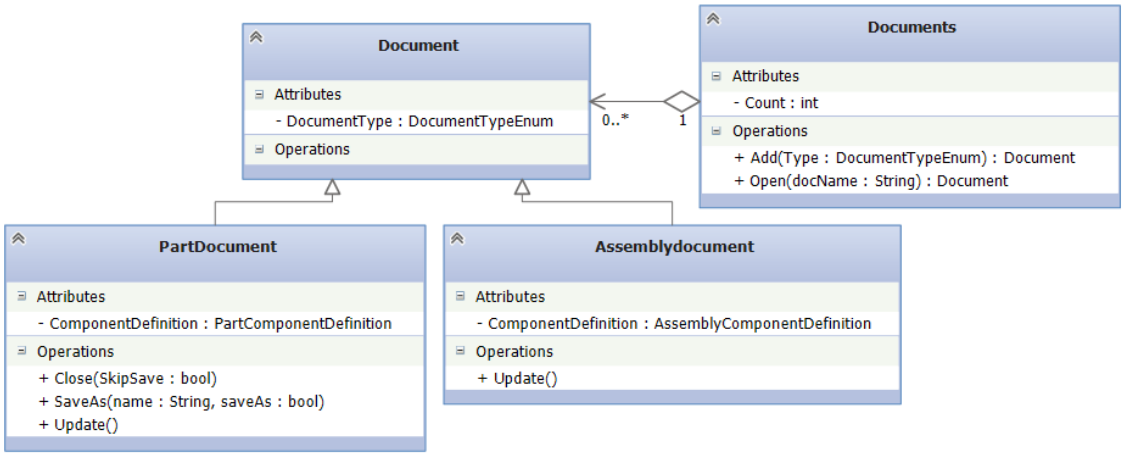
A continuació es farà una introducció de les classes utilitzades. Per més informació, veure l'API pròpia d'*Autodesk Inventor* (annex 14.2).

Application



A partir d'aquesta classe podem accedir a la jerarquia de classes. És l'encarregada de controlar tot el conjunt de classes de l'API. També permet configurar l'entorn i tots els paràmetres de la plataforma.

Documents, Document, Part Document i AssemblyDocument

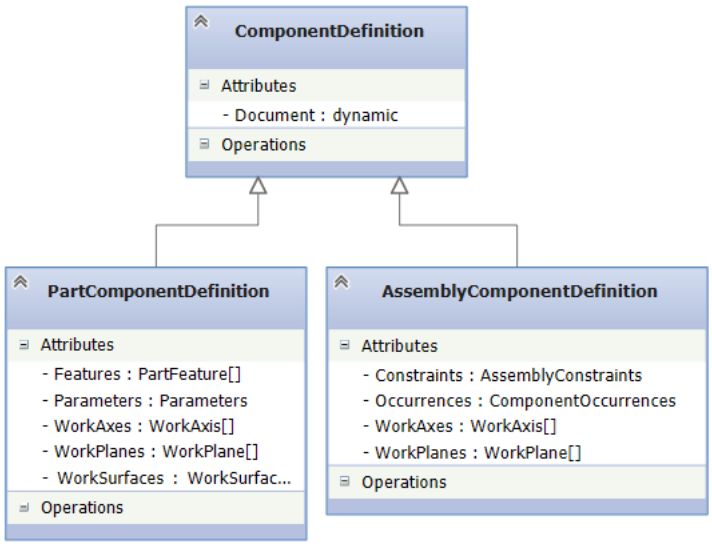


La classe Document conté tots els elements per generar elements de la plataforma. Tot hi haver quatre tipus de documents (PartDocument, AssemblyDocument, DrawingDocument i PresentationDocument), pel projecte només han estat necessaries les classes de PartDocument i AssemblyDocument.

El PartDocument s'encarrega de l'edició de peces simples, amb el qual podem definir les característiques, els valors de les mesures, les dimensions,... Representa un objecte individual, o peça, al qual podem editar les propietats a través de la classe PartComponentDefinition. Els fitxers amb els quals treballa tenen l'extensió .ipt.

El AssemblyDocument s'encarrega de l'edició d'un conjunt de peces com a tal. És a dir, és el responsable d'unir les peces i restringir els seus moviments. Per fer-ho, és necessari utilitzar el ComponentOccurrences, que instancia les peces creades amb el PartDocument. Un AssemblyDocument pot contenir altres AssemblyDocuments i PartDocuments repetits o no. Això vol dir que, si es modifiquen les propietats d'un PartDocument, totes les rèpliques iguals es modificaran de manera similar. Els fitxers amb els quals treballa tenen la extensió .iam.

ComponentDefinition, PartComponentDefinition i AssemblyComponentDefinition

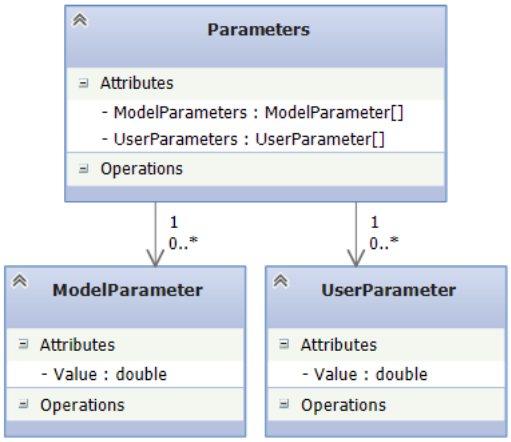


La classe ComponentDefinition i, per extenció, la classe PartComponentDefinition i AssemblyComponentDefinition, contenen la informació bàsica que defineix tant les peces com els assemblatges a què fan referència.

En aquest cas, la classe PartComponentDefinition dóna accés als Parameters, al PartFeature i als elements de l'espai de treball (punt d'origen, plans, eixos, punts i superfícies) per poder modificar les característiques que'ls componen.

La classe AssemblyComponentDefinition també dona accés als elements de l'espai de treball.

Parameters, UserParameter i ModelParameter



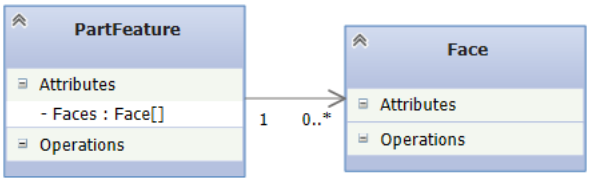
Bàsicament, aquesta classe conté accés als paràmetres que defineixen fòrmules o valors per dissenyar i dimensionar les diverses cotes de la peça. Per exemple, al

generar un cub, els paràmetres que tindrà aquest element seran: alçada, amplada i profunditat.

Podem distingir diversos tipus de paràmetres, però els utilitzats per el projecte en són dos:

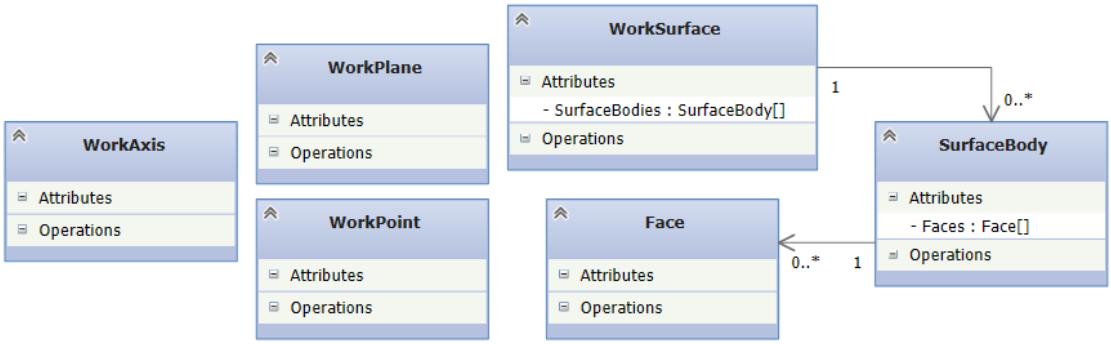
- UserParameter: conjunt de paràmetres afegits per l'usuari o amb el permís donat per l'aplicació per poder-los modificar dinàmicament.
- ModelParameter: tots els paràmetres de l'objecte.

PartFeature i Face



Les PartFeature contenen tots els elements que creen la peça referenciada. És a dir, contenen totes les cares, arestes, punts, eixos,... que compon una peça. Més concretament, l'aplicació només necessita accés a les cares de la peça.

WorkPoint, WorkAxis, WorkPlane, WorkSurface, SurfaceBody i Face



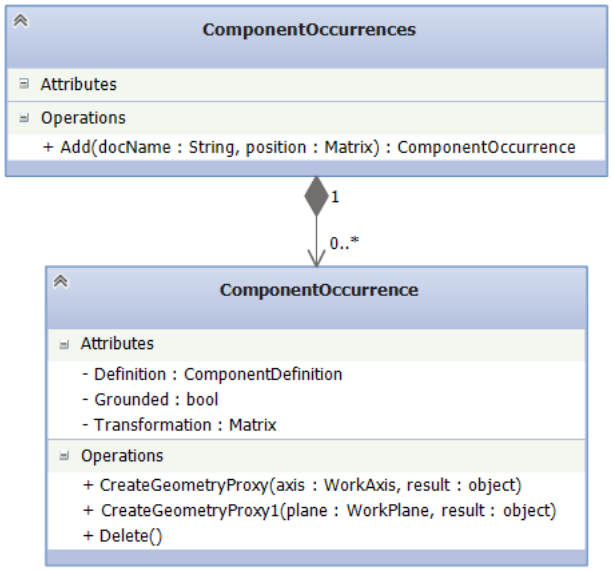
A partir del PartComponentDefinition i l'AssemblyComponentDefinition podem accedir als elements de referència dels objectes o de l'espai de l'assemblatge.

- WorkPoint: fa referència a punts de referència de peces o de l'assemblatge. Un punt per defecte que es genera és el punt corresponent a l'origen de l'espai de l'assemblatge o el punt de l'origen d'una peça.
- WorkAxis: equival al conjunt d'eixos direccionals de l'element. Per defecte corresponent als eixos de coordenades de la peça o de l'assemblatge, tot i que en poden haver-hi més.
- WorkPlane: per defecte, fa referència als plans que passen per l'origen de la peça o de l'assemblatge i coincideix amb dos eixos de coordenades.

Normalment hi ha els plans XY, YZ i XZ, tot i que en poden haver-hi més amb restriccions a altres elements de l'entorn.

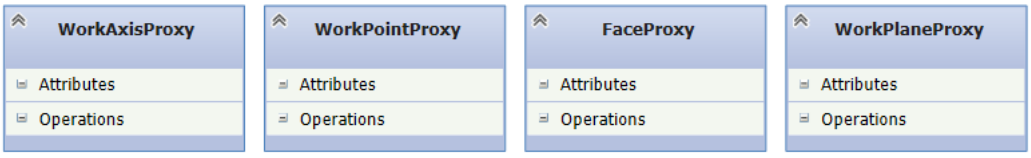
- WorkSurface: element que normalment correspont amb la superfície d'una cara, tot i que també pot seguir altres patrons establerts per l'usuari (per exemple, establir una superfície que divideixi un cub en dos o que la superfície correspongui amb el diàmetre que passa pel mig de les dents d'un engranatge). A partir d'aquesta classe, podem accedir al cos de superfície (SufaceBody) i obtenir la cara a què fa referència (Face).

ComponentOccurrences i ComponentOccurrence



Les ComponentOccurrences estan formades per la classe ComponentOccurrence. Una ComponentOccurrence és una representació lògica de les peces dintre dels assemblatges. En altres paraules, contenen la informació dels fitxers .ipt. A partir d'elles, podem accedir als PartDocument o generar Proxys dels elements que les componen.

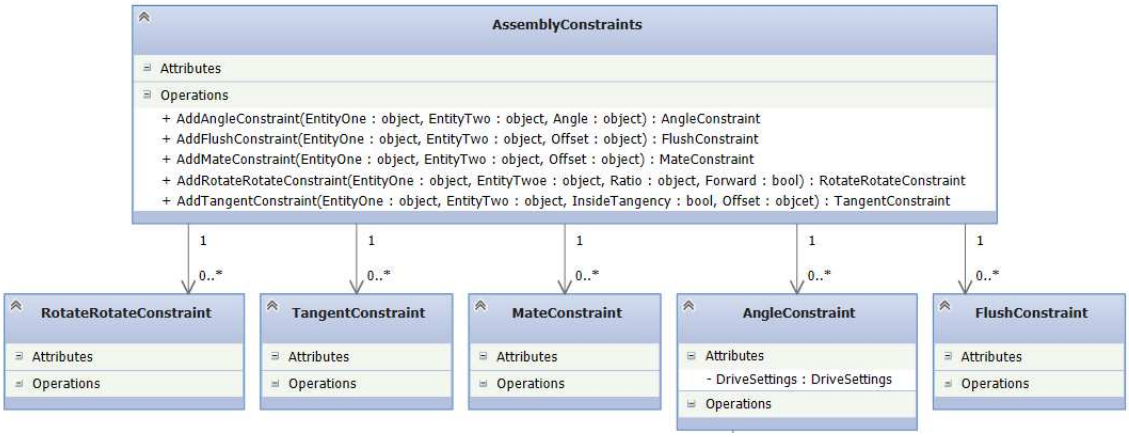
WorkAxisProxy, WorkPointProxy, FaceProxy i WorkPlaneProxy



La definició literal de l'ajuda d'*Autodesk Inventor* per a la definció d'aquests elements és: "*This is an assembly-context proxy object derived from its native definition-context object.*". En altres paraules, cada una d'aquestes classes correspon a una classe "representant" (*proxy*) per les seves classes natives, per exemple, el *WorkAxisProxy* representa a la classe *WorkAxis*.

La transformació dels elements de treball en representants és necessària perquè mètodes, com els mètodes de restriccions, puguin entendre amb quin element estan treballant.

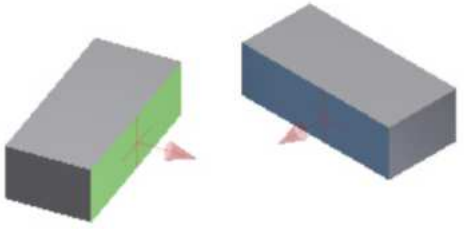
AssemblyConstraint, MateConstraint, FlushConstraint, TangentConstraint, AngleConstraint i RotateRotateConstraint



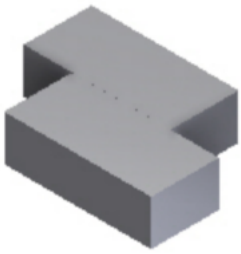
Als AssemblyConstraints s'hi accedeix a partir de l'AssemblyComponentDefinition, i consisteix en tot un conjunt de mètoed per afegir, modificar o eliminar els diversos tipus de restriccions que pot tenir el corresponent assemblatge. Pel projecte només serà necessari utilitzar els mètodes d'afegir constraints, que, per defecte, retornen una classe corresponent al constraint que es vol crear.

Els constraints utilitzats són:

- **MateConstraint:** La restricció d'encaix s'utilitza per aliniar elements propis dels objectes, tals com cares, eixos o arestes, entre dues peces indicades.

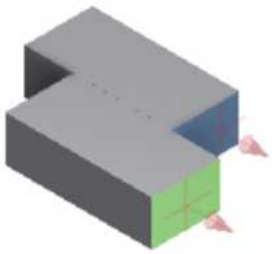


8.3.6a - Abans d'un MateConstraint

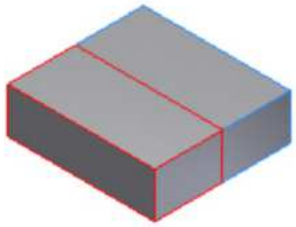


8.3.6b - Després d'un MateConstraint

- **FlushConstraint:** de manera similar al MateConstraint, consisteix en una restricció d'aliniament que s'encarrega d'anivellar elements dels objectes, tals com cares, eixos o arestes, entre dues peces indicades.

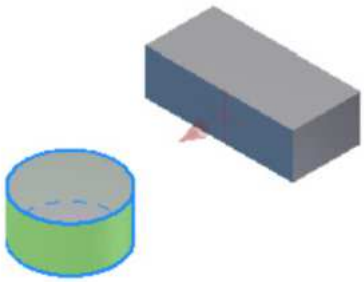


8.3.7a - Abans d'un FlushConstraint



8.3.7 b- Després d'un FlushConstraint

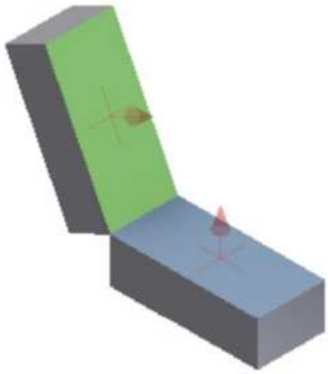
- **TangentConstraint:** Aquesta restricció s'utilitza per definir una relació tangencial entre dues peces. Normalent s'aplica entre una cara circular i una clara plana.



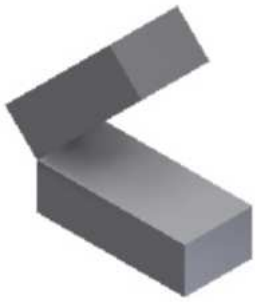
8.3.7a - Abans d'un TangentConstraint

8.3.7b - Després d'un TangentConstraint

- **AngleConstraint:** Una restricció angular és utilitzada per definir un angle d'obertura entre dues peces. Pot ser aplicat per cares, eixos o arestes, i es poden definir, donat eixos de referència, la direcció d'obertura dels angles.

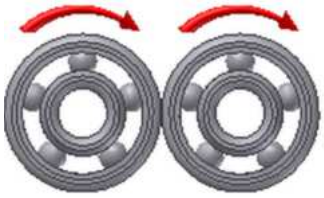


8.3.8a - Abans d'un AngleConstraint

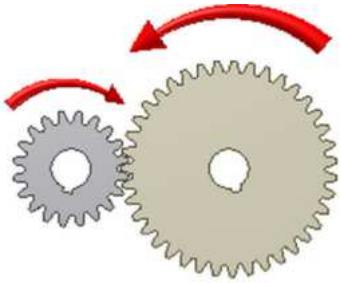


8.3.8b - Després d'un AngleConstraint

- **RotateRotateConstraint:** A diferència de les restriccions anteriors, aquesta restricció consisteix en establir un patró de moviment circular entre dues peces. El cas més utilitzat és, donada dues peces circulars restringides tangencialment entre elles, a l'aplicar un moviment circular sobre una d'elles, causa l'afecte de gir (invers o no) de la segona peça restringida. Es pot apreciar en la transmisió de moviment entre engranatges de l'aplicació.

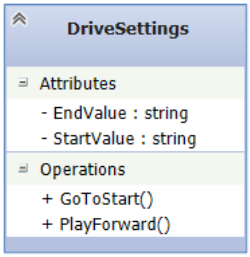


8.3.9a - Exemple de rotació en el mateix sentit



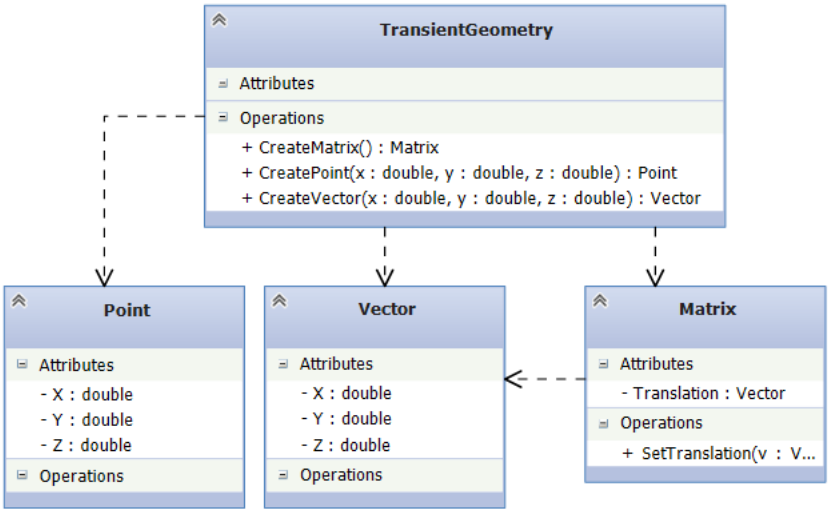
8.3.9b - Exemple de rotació inversa

DriveSettings



L'objecte DriveSetting proporciona accés a diverses configuracions de restriccions d'assemblatges i unions (*joints*). L'objectiu d'aquest objecte és modificar periòdicament alguns valors de la restricció a la qual forma part, causant l'efecte de moviment de la peça referenciada. En el projecte, per exemple, s'utilitza aquesta classe per simular el funcionament d'un motor a partir d'un AngleConstraint, al qual es va augmentant el valor de l'angle entre dos rangs preestablerts.

TransientGeometry, Point, Vector, Matrix



El concepte de TransientGeometry es pot interpretar de diverses maneres, tot i ser només dintre del contexte d'*Autodesk Inventor*. Tot i així, la definició més adequada per aquest projecte seria que el TransientGeometry és un objecte que cobreix diverses

construccions necessàries per dur a terme moltes tasques matemàtiques, més específicament tasques geomètriques, a través de l'API. Per exemple, els objectes de punt, vector i matriu.

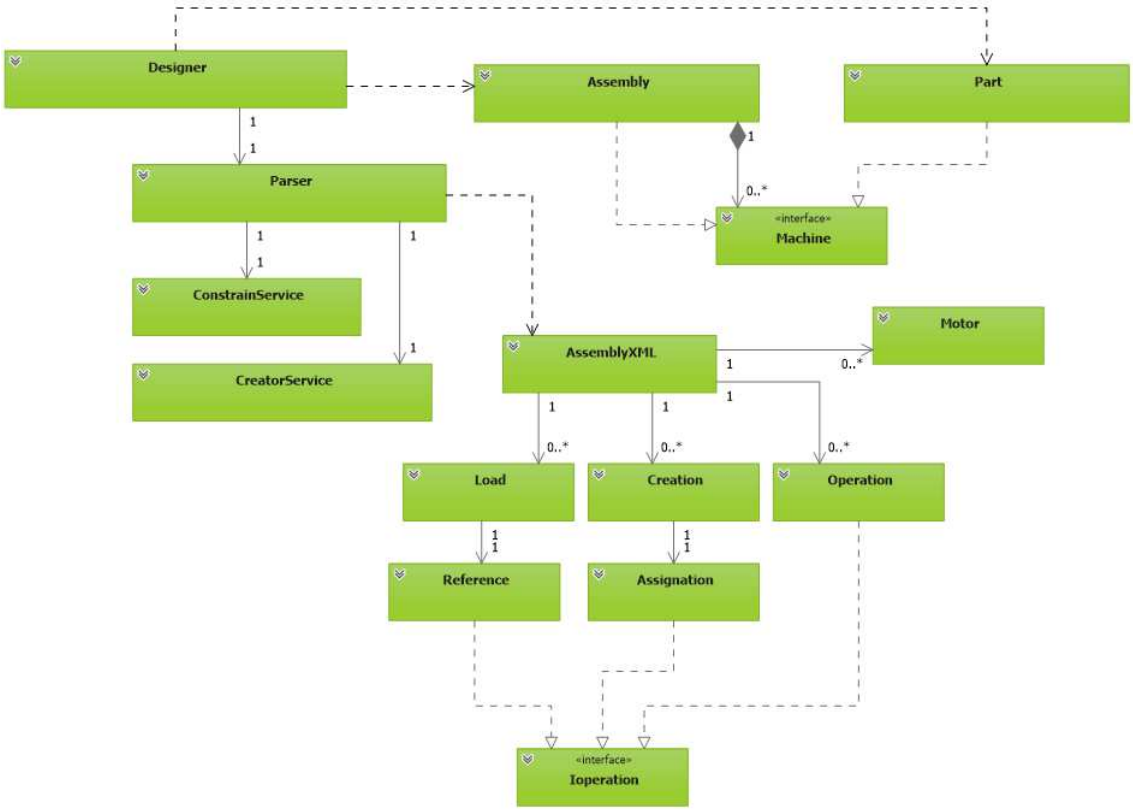
- **Point:** representa un punt geomètric dintre de l'espai de referència i posicionat amb els valors de X, Y i Z.
- **Vector:** representa un vector geomètric dintre de l'espai de referència amb les variables de X, Y, Z i longitud.
- **Matrix:** representa una matriu matemàtica de 4x4 amb diversos mètodes per transformar, rotar, transportar, pre-multiplicar, post-multiplicar,...

Aquestes classes són útils per generar nous elements dintre de l'espai de coordenades i, així, ajudar a restringir posicions i moviments dels objectes de l'assemblatge, gràcies a que les diverses restriccions poden també ser aplicades sobre aquests elements.

8.3.4 Mòdul d'aplicació

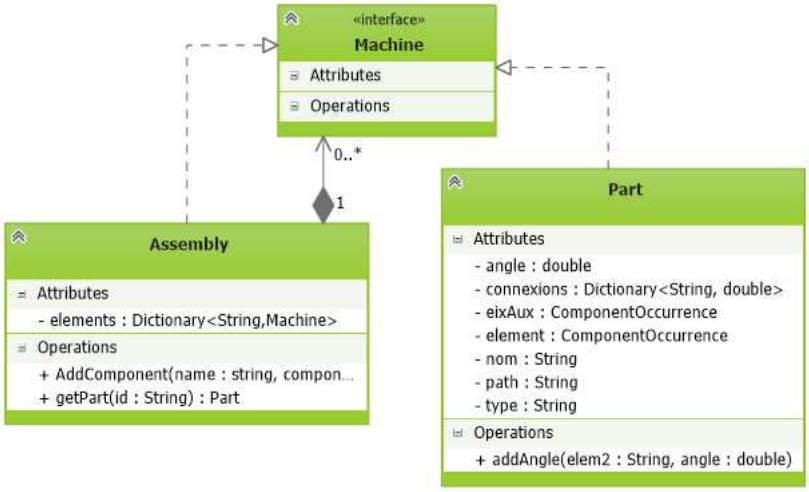
El mòdul d'aplicació és el principal mòdul del projecte, encarregat d'unir i utilitzar els mòduls de l'API d'*Autodesk Inventor* i de *System* amb l'objectiu d'aconseguir les funcionalitats desitjades del projecte. El procés principal que descriu aquest mòdul consisteix en parsejar el fitxer XML seleccionat per l'usuari, identificar el contingut i executar els mètodes corresponents per generar noves peces, restringir-les o assignar un motor, tal i com indiqui el contingut del fitxer. Un cop generat un primer assemblatge, podem optar per afegir nous assemblatges en el mateix espai de treball de l'*Autodesk Inventor* o iniciar la simulació del conjunt, sempre i quan s'hagi assignat un motor dintre el fitxer.

Per poder fer possible les funcionalitats esmentades, s'han necessitat crear les classes següents (veure figura 8.3.4.1).



8.3.4.1 - Diagrama de classes de l'aplicació

Machine Composite

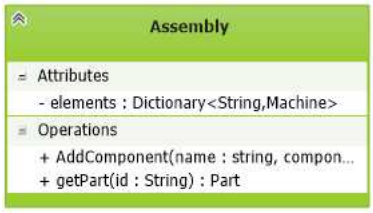


L'aplicació conté una estructura *composite* per tractar la jeraquia d'assemblatges carregades a partir dels fitxers XML d'entrada. Estem parlant d'un arbre, que correspont a una màquina *machine*, compost per peces mecàniques *Part* (engranatges, eixos,...) i assemblatges *Assembly* que, alhora, poden estar compostos per més peces i altres assemblatges.

El conjunt d'assemblatges i elements del simulador que es creïn es guardaran dintre una estructura *Assembly*.

La interfície *Machine* s' encarrega d'englobar tots els elements que componen una màquina, és a dir, objectes i assemblatges.

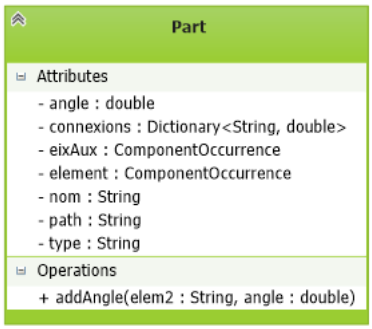
Assembly



La classe *Assembly* correspon, de manera visual o física, a un assemblatge. Conté un diccionari per identificar els assemblatges i peces que el forme a partir d'un nom. Els mètodes que el componen són:

- **AddComponent(name : string, component : Machine) -** Donat un objecte *Machine*, conegut amb un nom *name*, s'afageix com element de la classe Assembly actual. El resultat d'executar aquest mètode és guardar una entrada al diccionari amb el nom i l'element corresponent.
- **getPart(id : String) : Part -** Donat un nom *id*, que correspon a una peça, es va a buscar dintre de l'assemblatge actual un objecte *Part* amb el nom de *id.* Es pot donar el cas que el nom consisteixi en una adreça composta. Un exemple d'adreça composta seria: "assem1.assem2.nomPeça". Llavors, la peça amb nom nomPeça estarà a l'assemblatge assem2 que està dintre de l'assem1. En aquest cas, s'executarà de manera recursiva el mètode fins arribar a obtenir l'objecte dessitjat.

Part



La classe *Part* conté informació complementaria pels objectes creats als assemblatges. Els atributs que componen la classe són:

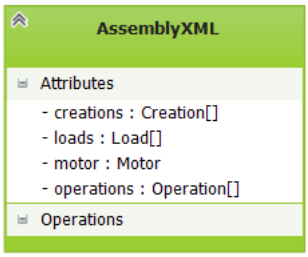
- **angle -** paràmetre de tipus real útil pel cas dels engranatges cònics, amb el qual podem determinar si és possible establir l'element que conté com a motor.

- **connexions -** diccionari, de claus amb el nom d'altres objectes *Part*, i valor de tipus real, que conté totes les posicions en graus relatives a ell.
- **eixAux** - referència a l'eix auxiliar, si en té, de tipus *ComponentOccurrence*. Només tindran valor els engranatges.
- **element** - referència a l'element que s'encapçula.
- **nom -** variable de tipus String que conté el nom de l'objecte dintre l'aplicació.
- **path -** direcció del sistema on es troba l'arxiu que conté l'objecte *element*.
- **type** - variable de tipus String amb la informació del tipus de l'element: *gear, shaft* o *bevel*.

I els mètodes que formen la classe són:

- **get, set** - metodes d'obtenció i modificació dels atributs
- **addAngle(elem : Stirng, angle : double) -** mètode per assigna la posició, en angle, de l'element indicat dintre el diccionari *connexions*.

AssemblyXML



La classe AssemblyXML conté l'estructura desitjada que han de tenir els fitxers XML per generar el mecanisme. És per aquesta raó que la classe AssemblyXML, i la resta de classes que utilitza, només contenen variables públiques, amb els mètodes de *get* i *set*, sense mètodes específics. Serà la classe XmlSerializer del mòdul *System* que especificarà l'estructura utilitzant aquesta classe.

Recordem que l'estructura d'aquests fitxers s'ha especificat en el punt *8.1 - Introducció al llenguatge XML*. Tot i així, podem veure a continuació un exemple del fitxer que l'aplicació espera rebre:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<assembly xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="assembly.xsd">
    //Carregar assemblatge 1 del primerExemple.xml (sense motor)
    <load>
        <name_as>as1</name_as>
        <dir_assembly>
```

```
            <dir>primerExemple.xml</dir>
            <use_motor>false</use_motor>
        </dir_assembly>
    </load>
    //Crear engranatge cònic 1
    <creation>
        <name>b1</name>
        <part_info>
            <part>new bevel</part>
            <param>30</param>
        </part_info>
    </creation>
    //Unir engranatge cònic 1 amb eix 1 de l'assemblatge 1 a 10 cm
    <operation>
        <constraint>join</constraint>
        <param>b1</param>
        <param>as1.a1</param>
        <param>10</param>
    </operation>
    //Engranatge 1 de l'assemblatge 1 amb funció de motor
    <motor>
        <motor_name>as1.g1</motor_name>
    </motor>
</assembly>
```

Cada classe i variables que s'explicaran a continuació contenen anotacions addicionals per cada etiqueta que pugui contenir els fitxers XML. Es destaca el nom de l'etiqueta que rebrà l'element dintre del fitxer. En primer lloc, la classe AssemblyXML correspondrà en el fitxer XML amb l'etiqueta ***<assembly>.*** L'ordre de les variables que ha de seguir l'estructura d'AssemblyXML és:

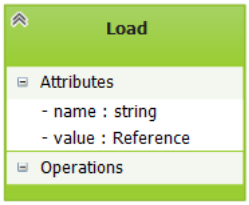
- **loads : Load[]** - Encapçulats amb l'etiqueta ***<load>***
- **creations : Creation[]** - Encapçulats amb l'etiqueta ***<creation>***
- **operations : Operation[]** - Encapçulats amb l'etiqueta ***<operation>***
- **motor : Motor** - Encapúlats amb l'etiqueta ***<motor>***

Ioperation

En diverses ocasions de les següents classes s'utilitzen classes que extenen de la interfície *Ioperation*. Aquesta interfície conté dues variables, les quals les funcionalitats varien segons la classe que l'extengui. En tot cas, les variables són:

- **opname : string -** paraula pensada per contenir un nom de referència.
- **parameters : string[] -** llista de paràmetres, utils per la classe on es troben.

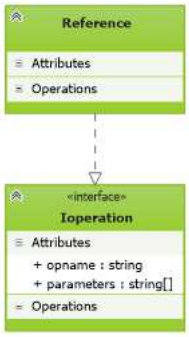
Load



La classe Load conté les variables necessàries per identificar i carregar altres fitxers XML per poder afegir sub-estructures a l'assemblatge arrel. Recordem que aquestes sub-estructures es guardaran dintre l'estructura *Composite*, en una classe *Assembly*. Per fer-ho, cal indicar les variables següents:

- **name : string -** Contindrà el nom que rebrà l'estructura que es carregarà. L'etiqueta corresponent és ***<name_as>***.
- **value : Reference -** Conté la informació pròpia de la classe Reference, que implementa la interfície *Ioperation*.

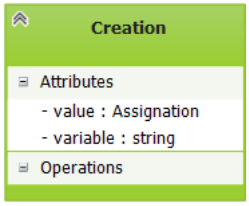
Reference



La classe Reference va encapçulada amb ***<dir_assembly>***. Les variables que conté són:

- **opname : string** - amb l'etiqueta ***<dir>***, conté l'adreça del fitxer a carregar.
- **parameters : string[]** - amb l'etiqueta ***<use_motor>***. Consisteix en un únic paràmetre que correspon en realitat a un booleà. Indica si es vol utilitzar com a motor principal el de l'assemblatge a carregar o el de l'etiqueta *<motor>* que es troba en el fitxer que s'està parsejant actualment.

Creation



La classe *Creation* conté les variables necessàries per generar una peça. Dintre l'etiqueta podem trobar:

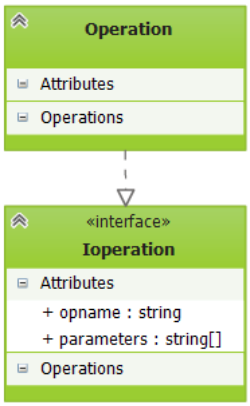
- **variable : string** - contindrà el nom que rebrà l'element a crear. L'etiqueta que l'engloba és ***<name>***
- **value : Assignation** - Conté la informació pròpia de la classe Assignation, que implementa la interfície Ioperation.

Assignation

La classe Assignation correspont a l'etiqueta ***<part_info>.*** Les variable que conté són:

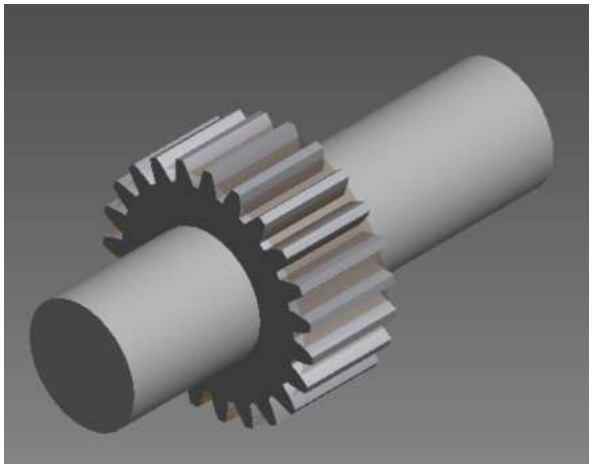
- **opname : string -** amb l'etiqueta ***<part>***. Conté el tipus d'element a crear. Actualment s'espera obtenir "new gear", "new axis" o "new bevel".
- **parameters : string[] -** amb l'etiqueta ***<param>***, consisteix en una llista de paràmetres segons el tipus d'element a crear.
    - Per un "new gear", s'espera obtenir el diàmetre.
    - Per un "new axis", s'espera obtenir el diàmetre de l'eix i la longitud.
    - Per un "new bevel", s'espera obtenir el número de dents.

Operation



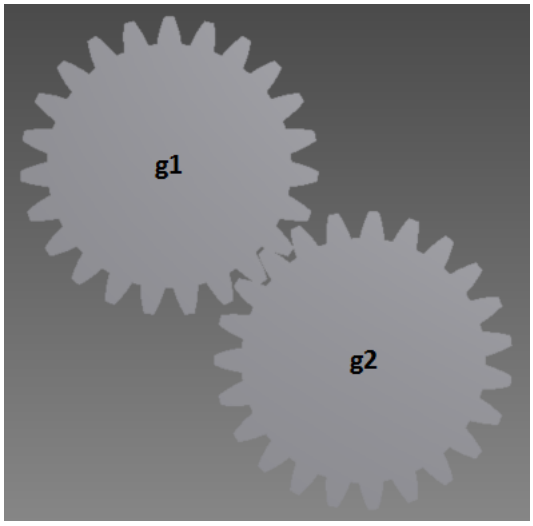
La classe Operation extén de la interfície Ioperation. Conté les variables necessàries per assignar una restricció entre dos elements. Dintre l'etiqueta podem trobar:

- **opname : string -** amb l'etiqueta ***<constraint>***. Conté el tipus de restricció a realitzar. Actualment només hi ha una restricció disponible, "join", però s'ha fet aquesta propietat pensant en un futur projecte.
- **parameters : string[]** - amb l'etiqueta ***<param>***, consisteix en la llista de noms dels elements els quals se'ls hi aplicarà la restricció.
    - En cas que un d'aquests elements fes referència a un eix, el tercer paràmetre faria referència al punt de la longitud de l'eix on s'unirà l'engranatge. En la imatge 8.3.4.2 veiem un engranatge unit amb un eix de 10 cm, on la base de l'engranatge coincideix amb la longitud central de l'eix (5 cm).



8.3.4.2 - Unió d'un eix amb un engranatge

    - En cas d'unir dos engranatges del mateix tipus, el tercer paràmetre conté el valor, en graus, de la posició del segon engranatge respecte del primer. Per exemple, a la imatge 8.3.4.3, l'engranatge g2 s'ha posicionat a 45 graus de l'engranatge g1.



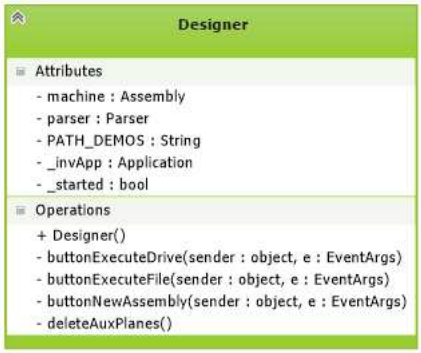
8.3.4.3 - Unió de g2 sobre g1 a 45 graus

Motor

La classe Motor s'utilitza per indicar, dintre de l'assemblatge, quin element farà la funció de motor. Disposa d'una única variable:

- **name : string** - amb l'etiqueta ***<motor_name>***, indica el nom de l'element a realitzar la funció de motor.

Designer



La classe Designer és la classe principal del projecte. És la classe on s'inicien tots els processos de l'aplicació, concretament, els d'inicialització i els indicats per l'usuari a través de la interfície.

A la llista de les seves variables distingim les constants, indicades en majúscules, i els següents atributs:

- **machine:** element de tipus *Assembly* que engloba l'estructura dels assemblatges carregats.
- **parser:** fa referència a la classe *Parser* explicada a continuació.
- **_invApp***:* correspont a l'aplicació d'*Autodesk Inventor*. A través d'ella podem accedir a tota l'API del mòdul d'Inventor.
- **_started:** atribut de tipus booleà que indica si l'aplicació d'*Autodesk Inventor* està oberta.

Els mètodes de la classe Designer corresponen en:

- ***void buttonExecuteFile(object sender, EventArgs e)*:** Obre una pantalla emergent per poder seleccionar el fitxer .xml desitgat. Un cop seleccionat, la classe de l'atribut *parser* generarà tot el contingut indicat al fitxer d'entrada i les restriccions pertinents. Per realitzar la seva tasca utilitzarà el mètode *parserFile* de la classe *Parser.*
- ***void buttonExecuteDriver(object sender, EventArgs e)***: Si es dóna el cas que ja s'ha cridat el *buttonExecuteFile,* al seleccionar aquest botó es posarà en moviment l'element indicat com a motor en el fitxer d'entrada. Si no n'hi

hagués, no faria res. En el cas de no haver carregat amb anterioritat cap fitxer, el *Designer* avisarà a l'usuari amb una pantalla emergent.

- ***void buttonNewAssembly(object sender, EventArgs e)***: La funció d'aquest botó és auxiliar. Serveix per obrir nous espais d'assemblatge dintre l'*Autodesk Inventor.* Útil per carregar diversos fitxers en espais diferents i sense reiniciar l'aplicació.

Parser

La classe *Parser* és la classe més completa de l'aplicació. La funció principal, com el nom indica, és parsejar el contingut del fitxer XML d'entrada, indicat per l'usuari, i executar les operacions necessàries per obtenir l'assemblatge desitjat. Aquesta classe utilitza les classes de *CreatorService,* encarregada en crear elements, i *ConstraintService*, encarregada en restringir, per aconseguir-ho.

Les variables que la formen són:

- **constrainer -** Objecte de tipus *ConstraintService*, per executar les restriccions entre peces.
- **creator** - Objecte de tipus *CreatorService*, per crear els elements a utilitzar.
- **_invApp***:* correspont a l'aplicació d'*Autodesk Inventor*. A través d'ella podem accedir a tota l'API del mòdul d'Inventor.

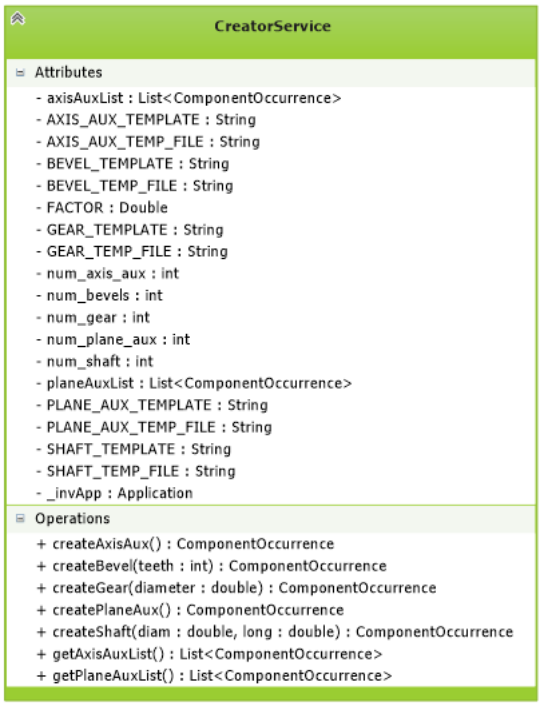
I els mètodes són:

- ***void Parser(Application app)*** - Constructor de la classe que inicialitza les variables.

- ***Assembly parserFile(String xml, String name, String dir, bool motor, String path_assem)***: llegeix un fitxer xml d'entrada aplicant l'estructura *AssemblyXML.* Si tot és correcte, executa les operacions necessàries per crear els objectes, aplicar operacions d'unió i assignar un motor.
- ***void loadModules(String dir, String path_assem, Assembly resultat, AssemblyXML assem):*** carrega els mòduls indicats per les etiquetes <*load*> del fitxer d'entrada.
- ***void createParts(String dir, Assembly resultat, AssemblyXML assem):*** executa les operacions per crear objectes basant-se amb les dades incloses a l'etiqueta <*creation*> del fitxer d'entrada.
- ***void applyOperations(Assembly resultat, AssemblyXML assem):*** executa els processos per aplicar les operacions d'unió, indicades per les etiquetes <*operation*> en el fitxer d'entrada.
- ***void applyMotorFunction(bool motor, Assembly resultat, AssemblyXML assem):*** executa els processos per assignar un motor a l'assemblatge que s'està construint, basant-se amb l'etiqueta <*motor*> del fitxer d'entrada.
- ***Part makeCreation(Object operation)***: Comprova que l'objecte rebut sigui de tipus *Assignation* (propi de l'*AssemblyXML*). Si és així, obté el nom de l'acció a realitzar (new gear, new axis o new bevel) i realitza les accions necessàries per generar un element del tipus indicat.
- ***void makeOperation(Object operation, Assembly assem)***: Comprova que el primer objecte rebut sigui de tipus *Operation* (propi de l'AssemblyXML). Si és així, obté els dos elements a restringir de l'assemblatge *assem* i els uneix segons el tipus dels elements (*gear-gear,axis-gear,bevel-bevel,axis-bevel*).
- ***void gearWithGear(Ioperation op, Part obj1, Part obj2):*** executa tot un conjunt de restriccions i operacions per fer possible la unió entre dues peces de tipus "engranatge cilíndric", basant-se amb la informació inclosa en l'objecte de tipus *Ioperation*.
- ***void gearWithShaft(Ioperation op, Part obj1, Part obj2):*** executa tot un seguit d'operacions i restriccions per fer possible la unió entre una peça de tipus "engranatge cilíndric", i una peça de tipus "eix de transmissió", basant-se amb la informació inclosa en l'objecte de tipus *Ioperation*.
- ***void bevelWithBevel(Ioperation op, Part obj1, Part obj2):*** executa tot un seguit d'operacions i restriccions per fer possible la unió entre dues peces de tipus "engranatge cònic", basant-se amb la informació inclosa en l'objecte de tipus *Ioperation*.
- ***void bevelWithShaft(Ioperation op, Part obj1, Part obj2):*** executa tot un seguit d'operacions i restriccions per fer possible la unió entre una peça de tipus "engranatge cònic", i una peça de tipus "eix de transmissió", basant-se amb la informació inclosa en l'objecte de tipus *Ioperation*.

- ***void assigMotor(Assembly assem, String id)*:** Donat un identificador d'un objecte propi de l'assemblatge *assem*, es genera una restricció d'angle sobre l'element per aconseguir la funció de motor.
- ***double getDiameterSpurGear(ComponentOccurrence occ):*** donada la referència (*ComponentOccurrence*) d'un element de tipus "engranatge cilíndric", es retornarà el valor del seu diàmetre.
- ***double margeDist(ComponentOccurrence axis, double dist):*** donat un *ComponentOccurrence* d'un eix de transmissió, comprova que la distància on es vol unir un altre component no superi la llargària del mateix eix. Si és així, retorna la distància màxima permesa, equivalent a la longitud de l'eix.
- ***void relateTeeth(ComponentOccurrence bevel1, ComponentOccurrence bevel2):*** Donada dos *ComponentOccurrences* d'engranatges cònics, es modificaran els paràmetres per relacionar-los entre si. En cas de no utilitzar aquest mètode, no s'assegura l'encaix entre els elements de tipus *bevel* que es vulguin unir.
- ***void makeTranslation(ComponentOccurrence gear1, ComponentOccurrence gear2, double posX, double posY):*** agafant com a punt de referència l'origen del primer *ComponentOccurrence*, es posiciona el segon a la posició (posX, posY, 0).
- ***void makeRotation(ComponentOccurrence gear1, double angle, double posX, double posY, double posZ):*** es fa rotar la referència *gear1*, un angle indicat, utilitzant com a referència un vector amb (posX, posY, posZ).
- ***void defineQuadrant(ComponentOccurrence g1, ComponentOccurrence g2, double angle, double dist):*** Es posicionarà l'element *g2* al centre d'un dels quatre quadrants del voltant de l'element *g1*. El quadrant es determinarà amb l'angle indicat, i la posició de les X i de les Y es determinaran amb la distància indicada.
- ***double getDiameterBevel(ComponentOccurrence occ)***: donat el *ComponentOccurrence* d'un engranatge cònic, retorna el valor del diàmetre.

CreatorService



La classe *CreatorService,* com el nom indica, s'encarrega d'aplicar les operacions necessàries per crear elements per l'assemblatge que s'està dissenyant. Els elements que s'obtenen com a resultat són de tipus *ComponentOccurrence*, propi de l'API d'*Autodesk Inventor*.

Distingim atributs en majúscules, que corresponent a constants. Les constants finalitzades amb **_FILE* contenen les adreces del directori on es troben els elements temporals; les finalitzades amb **_TEMPLATE* corresponen a les adreces on es guarden els fitxers dels elements per defecte. A més, hi ha una constant, *FACTOR*, que conté el valor per obtenir el nombre de dents dels engranatges cilíndrics.

A més de les constants, la classe conté els següents atributs:

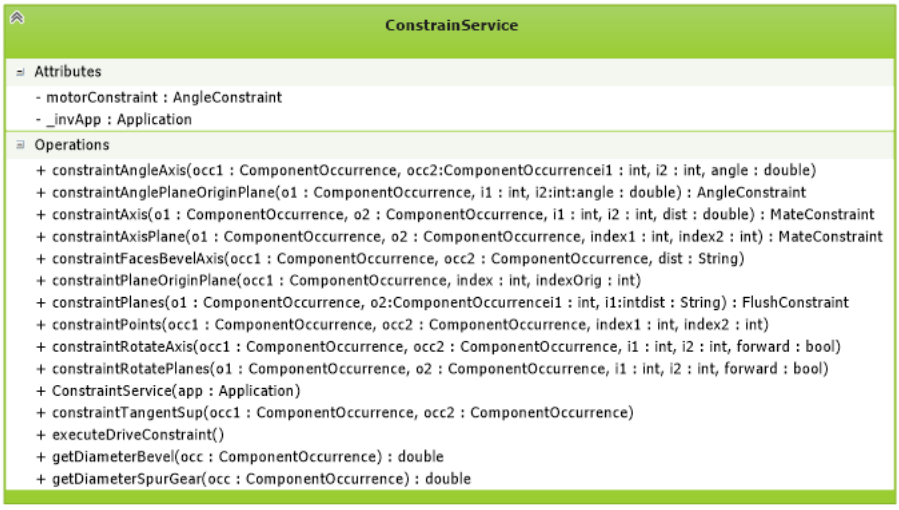
- **_invApp -** correspont a l'aplicació d'*Autodesk Inventor*. A través d'ella podem accedir a tota l'API del mòdul d'Inventor.
- **axisAuxList -** Llista, de tipus *ComponentOccurrence*, que conté tots els eixos auxiliars que s'han creat dintre l'assemblatge.
- **planeAuxList -** Llista, de tipus *ComponentOccurence,* que conté tots els plans auxiliars que s'han creat dintre l'assemblatge.
- **num_gears -** nombre d'engranatges cilíndrics creats.
- **num_shaft -** nombre d'eixos de transmissió creats.
- **num_bevels -** nombre d'engranatges cònics creats.
- **num_axis_aux -** nombre d'eixos auxiliars creats.

- **num_plane_aux -** nombre de plans auxiliars creats.

I els mètodes següents:

- ***CreatorService(Inventor.Application app):*** constructor de la classe *CreatorService*. En ell s'inicialitzen els atributs de la classe, com l'*_invApp*, que agafa el paràmetre *app.*
- ***ComponentOccurrence createGear(double diameter)***: Donat un diàmetre, es genera un engranatge cilíndric amb el diàmetre especificat.
- ***ComponentOccurrence createShaft(double diameter, double longitud)***: donat un diàmetre i una longitud, es genera un eix de transmissió utilitzant aquests valors.
- ***ComponentOccurrence createBevel(int teeth):*** donat un númer de dents, es genera un engranatge cònic amb el nombre de dents indicat.
- ***ComponentOccurrence createAxisAux():*** es genera un eix de coordenades per defecte.
- ***ComponentOccurrence createPlaneAux()***: es crea un pla de coordenades per defecte.
- ***List<ComponentOccurrence> getAxisAuxList()*:** es retorna la llista d'eixos auxiliars creats dintre l'assemblatge.
- ***List<ComponentOccurrence> getPlaneAuxList()*:** es retorna la llista de plans auxiliars creats dinte l'assemblatge.

ConstraintService



La classe *ConstraintService*, com el nom indica, s'encarrega d'aplicar restriccions dintre la plataforma d'*Autodesk Inventork*. Generalment les restriccions s'apliquen entre peces, o entre una peça i un element del sistema de coordenades de l'origen (eixos, plans,...).

Els atributs propis de la classe són:

- **_invApp -** correspont a l'aplicació d'*Autodesk Inventor*. A través d'ella podem accedir a tota l'API del mòdul d'Inventor.
- **motorConstraint -** Restricció, de tipus *AngleConstraint*, propi de l'API d'*Inventor*, que defineix el moviment del motor.

I els mètodes que la formen són:

- ***ConstraintService(Inventor.Application app)***: Constructor de la classe *ConstraintService*, on s'inicialitza l'atribut d'*_invApp* amb el paràmetre *app*.
- ***void constraintPlaneOriginPlane(ComponentOccurrence occ1, int index, int indexOrigin)***: Es crea una restricció de coincidència entre un pla de l'element *occ1,* i un pla de l'origen de coordenades. Els plans s'indiquen amb els índex especificats.
- ***void constraintPoints(ComponentOccurrence occ1, ComponentOccurrence occ2, int index1, int index2)***: Es crea una restricció de coincidència entre un pund e l'element *occ1*, i un punt de l'element *occ2*. Els punts s'indiquen amb els índex especificats.
- ***MateConstraint constraintAxis(ComponentOccurrence occ1, ComponentOccurrence occ2, int index1, int index2, double dist)***: Es crea una restricció de coincidènca, a una distància específica *dist,* entre un eix de l'element *occ1,* i un eix de l'element *occ2*. Els eixos s'indiquen amb els índex especificats.
- ***MateConstraint constraintAxisPlane(ComponentOccurrence occ1, ComponentOccurrence occ2, int index1, int index2):*** Es crea una restricció de coincidènca entre un eix de l'element *occ1* i un pla de l'element *occ2*. Els eixos s'indiquen amb els índex especificats.
- ***FlushConstraint constraintPlanes(ComponentOccurrence occ1, ComponentOccurrence occ2, int index1, int index2, string dist)***: Es crea una restricció de coincidènca, a una distància específica *dist,* entre un pla de l'element *occ1,* i un pla de l'element *occ2*. Els plans s'indiquen amb els índex especificats.
- ***void constraintTangentSup(ComponentOccurrence occ1, ComponentOccurrence occ2)***: Es genera una restricció tangencial entre superfícies dels elements *occ1* i *occ2*.
- ***void constraintAngleAxis(ComponentOccurrence occ1, ComponentOccurrence occ2, int index1, int index2, double angle)***: Es genera una restricció angular entre els eixos dels elements *occ1* i *occ2* amb un angle determinat. Els eixos s'esfecifíquen amb els índex donats.

- ***void constraintFacesBevelAxis(ComponentOccurrence occ1, ComponentOccurrence occ2, String dist)***: Es genera una restricció de coincidència entre la cara base externa d'un engranatge cònic *occ1* amb la cara base d'un eix de transmissió *occ2*.
- ***void constraintRotatePlanes(ComponentOccurrence occ1, ComponentOccurrence occ2, int index1, int index2, bool forward)***: Es genera una restricció de moviment entre dos plans dels elements *occ1* i *occ2*. El sentit del moviment es determina amb el booleà *forward* on, si és cert, els elements giraran en el mateix sentit. Els plans es determinen amb els índex especificats.
- ***void constraintRotateAxis(ComponentOccurrence occ1, ComponentOccurrence occ2, int index1, int index2, bool forward)***: Es genera una restricció de moviment entre dos eixos dels elements *occ1* i *occ2*. El sentit del moviment es determina amb el booleà *forward* on, si és cert, els elements giraran en el mateix sentit. Els eixos es determinen amb els índexs especificats.
- ***AngleConstraint constraintAnglePlaneOriginPlane(ComponentOccurrence occ1, int index1,int index2, double angle)*****:** Es genera una restricció angular entre un pla de l'element *occ1* amb un pla de l'origen, a un cert angle determinat. Els plans es determinen amb els índexs donats. El resultat és un *AngleConstraint,* al qual se li pot aplicar el *Drive Constraint*, encarregat d'incrementar l'angle de la restricció.
- ***double getDiameterSpurGear(ComponentOccurrence occ)***: igual com el mètode del mateix nom inclòs a la classe *CreateService*, donat el *ComponentOccurrence* d'un engranatge cilíndric, retorna el valor del diàmetre.
- ***double getDiameterBevel(ComponentOccurrence occ)***: igual com el mètode del mateix nom inclòs a la classe *CreateService*, donat el *ComponentOccurrence* d'un engranatge cònic, retorna el valor del diàmetre.
- ***void executeDriveConstraint()***: Executa el *Drive Constraint* a l'atribut *motorConstraint*, que consisteix en una restricció de tipus *AngleConstraint.*

9. Implementació i proves

A continuació es detallarà el procés que segueix el projecte per generar un assemblatge a partir d'un fitxer .xml, donant èmfasis als problemes trobats i les solucions o alternatives proposades.

9.1 Fitxer XML

El treball previ, abans d'executar el projecte, és obtenir un fitxer en format XML amb els mòduls, les peces, les unions de les peces i el motor que volem a l'assemblatge.

9.1.1 XML basat en XML Schema

A més de tenir definit dintre el codi l'estructura que s'espera al rebre un fitxer XML (estructura definida per l'*AssemblyXML*), es definirà l'esquema en un fitxer XSD extern.

El sistema desenvolupat treballa sobre XML Schema, que descriu l'estructura i les restriccions de documents XML de manera molt precisa, més enllà de les normes sintàctiques imposades pel propi llenguatge XML. El llenguatge està basat en el sistema d'etiquetes utilitzat pel propi XML. Aconsegueix una percepció del document amb un alt nivell d'abstracció que defineix:

- elements que poden apareixer.
- atributs que poden apareixer.
- quina herència té cada element.
- el número de fills permesos.
- si un element pot està buit o ser null.
- el tipus esperat de cada element.
- valors per defecte d'elements i atributs.

L'estructura que seguirà el fitxer XSD dels arxius emprats pel projecte és el següent:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="assembly">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="load">
     <xs:complexType>
      <xs:sequence>
       <xs:element type="xs:string" name="name_as"/>
       <xs:element name="dir_assembly">
        <xs:complexType>
         <xs:sequence>
          <xs:element type="xs:string" name="dir"/>
          <xs:element type="xs:string" name="use_motor"/>
         </xs:sequence>
```

```
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
      <xs:element name="creation" maxOccurs="unbounded" minOccurs="0">
       <xs:complexType>
        <xs:sequence>
         <xs:element type="xs:string" name="name"/>
         <xs:element name="part_info">
          <xs:complexType>
           <xs:sequence>
            <xs:element type="xs:string" name="part"/>
            <xs:element type="xs:float" name="param" maxOccurs="unbounded" minOccurs="0"/>
           </xs:sequence>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
      <xs:element name="operation">
       <xs:complexType>
        <xs:sequence>
         <xs:element type="xs:string" name="constraint"/>
         <xs:element type="xs:string" name="param" maxOccurs="unbounded" minOccurs="0"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
      <xs:element name="motor">
       <xs:complexType>
        <xs:sequence>
         <xs:element type="xs:string" name="motor_name"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
</xs:schema>
```

Inicialment podem veure la declaració de la versió XML que s'utilitza, i tot seguit comença l'estructura pòpia del XML Schema. Distingim les següents etiquetes:

- **xs:schema -** Indica l'inici i el final de l'estructura XML Schema.
    - **xmlns:xs="http://www.w3.org/2001/XMLSchema" -** És un atribut pròpi de l'etiqueta schema. Fa referència a l'estàndard *Worldwide Web Consortioum* (w3c), una recomenació utilitzada per la majoria dels esquemes XML.
- **xs:element -** Indica el contingut d'un bloc. Normalment va acompanyat d'atributs, com per exemple:
    - **name="" -** Indica el nom que rebrà l'element.

- **maxOccurs="unbounded" -** Indica el número màxim de repeticions que pot tenir l'element. En aquest cas, pot ser ilimitat.
- **minOccurs="0" -** Indica el número mínim de repeticions que pot tenir l'element. En aquest cas, pot ser null.
- **type="xs:string" -** Indica el tipus de contingut com a String.
- **type="xs:float" -** Indica el tipus de contingut com a número real.

- **xs:complexType -** Ens informa que el contingut seguent és complex o està compost.
- **xs:sequence -** Ens informa que el contingut que vingui a continuació estarà inclós dintre una llista o seqüència.

A continuació es descriurà el patró a més alt nivell que pot interpretar l'aplicació:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<assembly xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                    xsi:noNamespaceSchemaLocation="assembly.xsd">
        <load>
                <name_as>...</name_as>
                <dir_assembly>
                        <dir>...</dir>
                        <use_motor>...</use_motor>
                </dir_assembly>
        </load>
            ...
        <creation>
                <name>...</name>
                <part_info>
                        <part>...</part>
                        <param>...</param>+
                </part_info>
        </creation>
            ...
        <operation>
                <constraint>...</constraint>
                <param>...</param>
                <param>...</param>
                <param>...</param>
        </operation>
            ...
        <motor>
                <motor_name>...</motor_name>
        </motor>
</assembly>
```

Tot i que l'estructura general s'ha definit amb anterioritat (veure apartat 8.1), cal destacar alguns matissos de cada bloc.

Assembly

Dintre l'etiqueta *assembly* s'hi afageix un nou atribut per indicar el fitxer XSD amb l'estructura XML Schema a seguir: ***xsi:noNamespaceSchemaLocation.*** L'etiqueta *assembly* és indispensable i única pel fitxer XML que llegirà l'aplicació.

Load

```
<load>
        <name_as>...</name_as>
        <dir_assembly>
                <dir>...</dir>
                <use_motor>...</use_motor>
        </dir_assembly>
</load>
```

L'etiqueta *load* s'encarrega de carregar sub-assemblatges d'altres fitxers XML. Està compost per una etiqueta *name_as*, que indica el nom que rebrà internament el sub-assemblatge que es generarà, i l'etiqueta *dir_assembly*, que indica l'adreça relativa del fitxer XML a carregar (etiqueta *dir*), i un booleà (etiqueta *use_motor*) indicant si s'utilitzarà el motor del modul a carregar o, en cas contrari, el motor indicat en el mateix fitxer, si n'hi ha.

Les etiquetes són necessàries i úniques dintre el mòdul *load*, no com la pròpia etiqueta *load* que és indispensable dintre l'etiqueta *assembly*, tot i poder-se repetir.

Creation

```
<creation>
        <name>...</name>
        <part_info>
                <part>...</part>
                <param>...</param>+
        </part_info>
</creation>
```

L'etiqueta *creation* s'encarrega de generar components per a l'assemblatge. Dintre l'etiqueta *assembly* es poden declarar diversos mòduls *creation*, o no declarar-ne cap.

Està composta per una etiqueta *name*, que contindrà el nom intern de la peça, i per una etiqueta composta *part_info*, amb tota la informació per poder generar un nou element. Aquesta informació conté una etiqueta *part*, per indicar què es vol construir, i un seguit d'etiquetes *param*, amb un nombre variable segons l'element a construir. A continuació es pot veure una taula amb les diverses opcions que accepta l'etiqueta *part* i els paràmetres que esperen rebre cada una d'elles.

part	*Número de paràmetres*	*Ús*
new gear	Un paràmetre	Diàmetre intern de l'engranatge
new axis	Dos paràmetres	1- Diàmetre d'amplada de l'eix 2- Longitud de l'eix
new bevel	Un paràmetre	Número de dents de l'engranatge cònic

Totes les etiquetes i el nombre de paràmetres són necessaris.

Operation

```
<operation>
        <constraint>...</constraint>
        <param>...</param>
        <param>...</param>
        <param>...</param>
</operation>
```

L'etiqueta *operation* s'encarrega d'aplicar les restriccions necessàries sobre els elements indicats per poder unir les peces de la manera esperada. Poden no haver etiquetes *operation* dintre el mòdul d'*assembly*.

Aquesta etiqueta està composta per l'etiqueta *constraint*, que indica l'operació a realitzar, i tot un seguit de paràmetres dintre les etiquetes *param*. A dia d'avui només existeix un tipus de restricció esperada: *join*. Tot i així, l'etiqueta *constraint* es manté per possibles futures ampliacions. Les etiquetes *param* indiquen, seguint l'ordre, dues peces a unir i un tercer valor necessari per poder realitzar la restricció.

A continuació es pot veure una taula que indica les diverses combinacions de peces que es poden unir, i l'ús del tercer paràmetre esperat (l'ordre dels dos primers elements no afecta la definició d'ús del tercer paràmetre).

Peça 1	*Peça 2*	*Ús del paràmetre adicional*
Engranatge cilíndric	Engranatge cilíndirc	Angle en graus de la peça 2 respecte la peça 1 en sentit horari
Engranatge cilíndric	Eix	Distància en cm entre la base de l'eix i la cara de l'engranatge
Eix	Engranatge cònic	Distància en cm entre la base de l'eix i la cara exterior de l'engranatge
Engranatge cònic	Engranatge cònic	Angle en graus de la peça 2 respecte la peça 1 en sentit horari

Motor

```
<motor>
        <motor_name>...</motor_name>
</motor>
```

L'etiqueta *motor* determina quin element farà la funció de motor dintre el bloc de l'*assembly*. Aquesta etiqueta és única, però pot no està inclosa al fitxer.

Aquest bloc està compost per una única etiqueta *motor_name* que conté el nom intern de l'element que farà la funció de motor. El nom pot fer referència a qualsevol tipus d'objecte ja generat.

9.1.2 Resultats esperats

Tot seguit podem veure els resultats esperats en unir totes les combinacions de peces explicades en el mòdul *operation*.

Tots els engranatges cilíndrics utilitzats (amb el nom g1 i g2) tenen el mateix diàmetre de 5 cm.

```
<creation>
        <name>g1</name>
        <part_info>
                <part>new gear</part>
                <param>5</param>
        </part_info>
</creation>
```

Tots els engranatges cònics utilitzats (amb el nom b1 i b2) tenen el mateix nombre de dents: 30 dents.

```
<creation>
        <name>b1</name>
        <part_info>
                <part>new bevel</part>
                <param>30</param>
        </part_info>
</creation>
```

Tots els eixos de transmissió utilitzats (amb el nom a1) tenen diàmetre de 2.5 cm i longitud de 10 cm.

```
<creation>
        <name>a1</name>
        <part_info>
                <part>new axis</part>
                <param>2.5</param>
                <param>10</param>
        </part_info>
</creation>
```

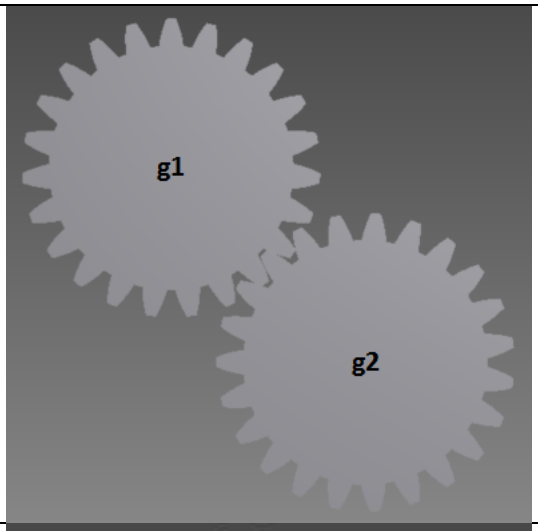
Per no mostrar tot el contingut dels fitxers, només s'indicaran els blocs de les operacions a aplicar, considerant que la creació de les peces utilitzades són les anteriors, que no es carreguen mòduls addicionals i que no s'aplica cap funció de motor.

Per tant, els resultats són els següents:

<table>
<tr>
<td>

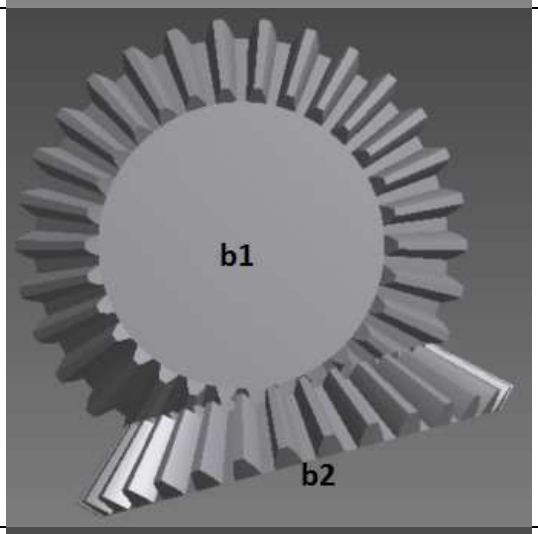
```
<operation>
        <constraint>join</constraint>
        <param>g1</param>
        <param>g2</param>
        <param>45</param>
</operation>
```

Nota: cal recordar que el sentit de l'angle és negatiu a partir de l'eix de les X.

</td>
<td> 
</td>
</tr>
<tr>
<td>

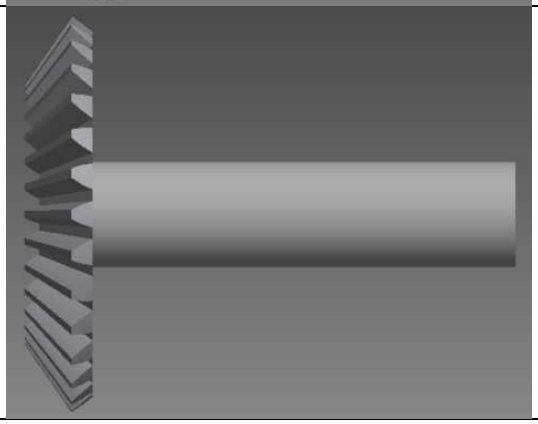
```
<operation>
        <constraint>join</constraint>
        <param>b1</param>
        <param>b2</param>
        <param>75</param>
</operation>
```

Podem veure que el b2 està a 75 graus respecte el b1 en sentit negatiu

</td>
<td> 
</td>
</tr>
<tr>
<td>

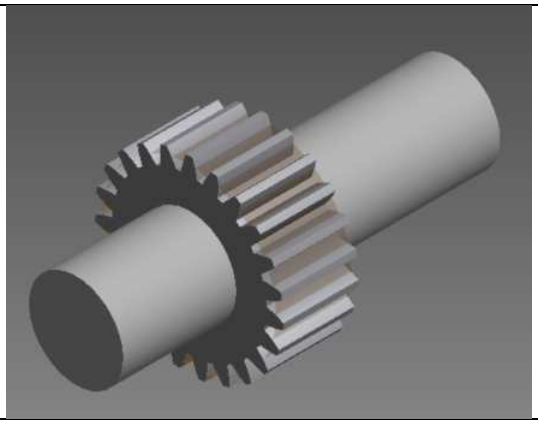
```
<operation>
        <constraint>join</constraint>
        <param>a1</param>
        <param>b1</param>
        <param>0</param>
</operation>
```

Al tenir distància 0, la base de l'eix s'uneix amb la base de l'engranatge cònic

</td>
<td>  </td>
</tr>
</table>

```
<operation>
        <constraint>join</constraint>
        <param>g1</param>
        <param>a1</param>
        <param>5</param>
</operation>
```

Podem veure que la base de l'engranatge queda just al mig de l'eix



9.2 Assemblador

Un cop tenim fitxers XML dels assemblatges desitjats, cal executar el programa assemblador per poder-los generar. El programa segueix un seguit de processos per poder aconseguir assemblar, creant els elements i les unions indicats en els fitxers d'entrada.

9.2.1 Inicialització de l'aplicació

Les llibreries de totes les classes que l'aplicació requereix per poder funcionar són les següents:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using System.IO;
using System.Xml;
using System.Xml.Serialization;
using System.Xml.Schema;
using Inventor;
```

Les constants de totes les classes que componen l'assemblador són les següents:

```
//Designer
private static String PATH_DEMOS = "../Demos";
//CreatorService
private static String GEAR_TEMP_FILE = "C:/Assembler/Temp/Gear_";
private static String SHAFT_TEMP_FILE = "C:/Assembler/Temp/Shaft_";
private static String BEVEL_TEMP_FILE = "C:/Assembler/Temp/Bevel_";
private static String AXIS_AUX_TEMP_FILE = "C:/Assembler/Temp/Aux_axis_";
private static String PLANE_AUX_TEMP_FILE = "C:/Assembler/Temp/Aux_plane_";
private static String GEAR_TEMPLATE = "C:/Assembler/Templates/Gear_template.ipt";
private static String SHAFT_TEMPLATE = "C:/Assembler/Templates/Shaft_template.ipt";
private static String BEVEL_TEMPLATE = "C:/Assembler/Templates/Bevel_template.ipt";
private static String AXIS_AUX_TEMPLATE =
                                "C:/Assembler/Templates/Aux_axis_template.ipt";
private static String PLANE_AUX_TEMPLATE =
```

```
                                    "C:/Assembler/Templates/Aux_plane_template.ipt";
private static Double FACTOR = 5;
```

I les variables locals de les diverses classes són:

```
//Comuna per a totes les classes
Inventor.Application _invApp;
//Designer
bool _started = false;
Parser parser;
Assembly machine;
//Parser
private CreatorService creator;
private ConstraintService constrainer;
//CreatorService
int num_gears = 0;
int num_shaft = 0;
int num_bevels = 0;
int num_axis_aux = 0;
int num_plane_aux = 0;
List<ComponentOccurrence> axisAuxList = new List<ComponentOccurrence>();
List<ComponentOccurrence> planeAuxList = new List<ComponentOccurrence>();
//ConstraintService
AngleConstraint motorConstraint = null;
```

Distingim els atributs pròpis de cada classe per poder fer funcionar l'Inventor, els registres d'objectes creats, l'assemblatge generat, les llistes d'elements auxiliars i la restricció angular amb funció de motor.

El desenvolupament es va realitzar dintre una única classe que, amb el temps, es va subdividir en diverses classes aconseguint més claredat en el codi, separació de les funcionalitats principals i millor gestió. El resultat va ser aconseguir les classes *Parser, CreatorService* i *ConstraintService*, separades de la classe *Designer*.

9.2.2 Classe Designer

La classe *Designer* és la classe on s'executen les funcionalitats dels botons de la interfície d'usuari. A partir d'ella, es criden la resta de classes.

El primer procés que segueix l'aplicació és inicialitzar algunes d'aquestes variables, principalment la variable _invApp, i obrir el menú de funcionalitats pròpia de l'aplicació.

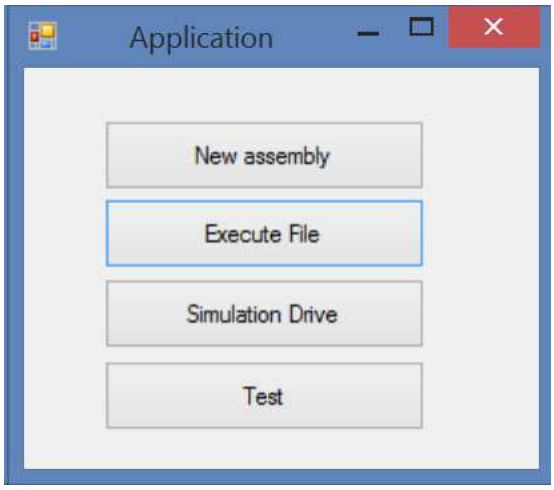
```
public Designer() {
   InitializeComponent();

   //Open Inventor. If not started, it will be
   try {
      _invApp= (Inventor.Application)Marshal.GetActiveObject("Inventor.Application");
   }
   catch (Exception) {
      try {
         Type invAppType = Type.GetTypeFromProgID("Inventor.Application");
         _invApp = (Inventor.Application)System.Activator.CreateInstance(invAppType);
         _invApp.Visible = true;
```

```
            _started = true;
        }
        catch (Exception ex2) {
            MessageBox.Show(ex2.ToString());
            MessageBox.Show("Unable to get or start Inventor");
        }
    }

    //Open automatically a new assembly
    _invApp.Documents.Add(DocumentTypeEnum.kAssemblyDocumentObject);

    parser = new Parser(_invApp);
}
```

Com podem veure, inicialment s'executarà la funció *InitializeComponent()*, mètode propi del Visual Studio, autogenerat al dissenyar la interfície d'usuari. Internament s'inicialitzen els botons i les propietats i elements de la finestra.

Tot seguit, s'intenta obtenir la referència pròpia de l'*Autodesk Inventor* amb la qual poder-hi treballar. Si ja hi hagués alguna referència, és a dir, que l'*Autodesk Inventor* ja estigués obert, no saltaria cap excepció i el procés no entraria en el primer "catch". Si no fós així i entrés, s'executa una crida per obrir i inicialitzar l'aplicació d'*Autodesk Inventor.* El resultat esperat és poder assignar la variable *_invApp*. Per acabar, s'obre un nou assemblatge buit a l'aplicació amb el qual poder començar a treballar i es crea una nova classe *Parser* assignada a la variable corresponent.

El resultat esperat a l'inicialitzar l'aplicació és tenir obert l'*Autodesk Inventor* amb un nou fitxer d'assemblatge buit i que aparegui una finestra amb diversos botons, com la següent:



Acte següent, l'aplicació espera que l'usuari pulsi algun dels botons.

9.2.3 Primer botó: *New assembly*

L'acció d'aquest botó és molt simple, amb una funcionalitat auxiliar. L'objectiu és obrir un nou fitxer d'assemblatge *.iam* dintre l'aplicació d'*Autodesk Intentor,* de manera similar a l'inicialització del programa. El motiu per haver afegit aquest botó és poder executar diversos fitxers XML d'entrada sense la necessitat de reiniciar el programa i tenir els assemblatges resultants separats en diversos fitxers.

```
private void buttonNewAssembly (object sender, EventArgs e) {
   try {
      _invApp.Documents.Add(DocumentTypeEnum.kAssemblyDocumentObject);
   }
   catch (Exception ex) {
      System.Console.WriteLine(ex.Message);
   }
}
```

9.2.4 Segon botó: *Execute File*

La funcionalitat d'aquest botó és la principal de tot el projecte. S'encarrega de seleccionar un fitxer XML, parsejar-lo i executar les accions necessaries per carregar altres moduls de l'assemblatge, si n'hi ha, generar les peces desitjades, restringir-les i assignar un motor, si es vol.

9.2.4.1 Procés *Execute File:* Botó d'execució del fitxer

Amb tot, el procés comença, dintre de la classe *Designer*, de la següent manera:

```
private void buttonExecuteFile(object sender, EventArgs e) {
   try {
      if (_invApp.Documents.Count == 0) {
         MessageBox.Show("Need to open an Assembly document");
         return;
      }
      if (_invApp.ActiveDocument.DocumentType !=
                                    DocumentTypeEnum.kAssemblyDocumentObject) {
         MessageBox.Show("Need to have an Assembly document active");
         return;
      }

      Stream myStream = null;
      OpenFileDialog openFileDialog1 = new OpenFileDialog();
      openFileDialog1.InitialDirectory = path_demos;
      openFileDialog1.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
      openFileDialog1.FilterIndex = 2;
      openFileDialog1.RestoreDirectory = true;
      if (openFileDialog1.ShowDialog() == DialogResult.OK) {
         try {
            if ((myStream = openFileDialog1.OpenFile()) != null) {
               String path_assem =
               System.IO.Path.GetDirectoryName(openFileDialog1.FileName) + "\\";
               using (myStream) {
                  machine = parser.parserFile(openFileDialog1.FileName, "Arrel",
                                                          "", true, path_assem);
```

```
                }
            }
        }
        catch (Exception ex) {
            MessageBox.Show("Error: Could not read file from disk. Original
                                                    error: " + ex.Message);
        }
    }
}
catch (Exception ex) {
    System.Console.WriteLine(ex.Message);
}
}
```

S'executen uns controls inicials sobre l'*Inventor* i es genera un *OpenFileDialog* encarregat d'obrir una nova finestra amb la funció de seleccionar fitxers. Si el fitxer seleccionat és l'esperat i tot ha sortit correctament, s'executarà el mètode *parserFile* amb el fitxer indicat. El resultat d'aquest mètode és assignar a la variable *machine* l'estructura lògica indicada amb el contingut del fitxer.

9.2.4.2 - Procés *Execute File*: parsejar el fitxer

```
public Assembly parserFile(String xml, String name, String dir, bool motor, String
                                                                    path_assem) {
        Assembly resultat = new Assembly();

        try {
                TextReader reader = new StreamReader(xml);
                XmlSerializer serializer = new XmlSerializer(typeof(AssemblyXML));
                AssemblyXML assem = (AssemblyXML)serializer.Deserialize(reader);

                loadModules(dir, path_assem, resultat, assem);

                createParts(dir, resultat, assem);

                applyOperations(resultat, assem);

                applyMotorFunction(motor, resultat, assem);

                reader.Close();
        }
        catch (Exception e) {
                System.Console.WriteLine(e.Message);
        }
        return resultat;
}
```

Dintre el mètode *parserFile*, de la classe *Parser,* es llegeix el fitxer XML indicat i s'obtenen les classes equivalents. Tot seguit, s'executen els sub-mètodes privats *loadModules, createParts, applyOperations* i *applyMotorFunction*. Tots quatre mètodes segueixen el mateix patró:

- Obtenir la llista desitjada de l'*AssemblyXML assem*.
- Si la llista no està buida, cal recorre-la.
- Per cada element de la llista, aplicar els mètodes desitjats.

El mètode *loadModules* identifica amb antel·lació si es carregarà un mòdul amb motor o sense. Aquesta comprovació es realitza llegint el valor del primer paràmetre inclòs en la informació del mòdul. Acte seguit, es crida de manera recursiva el mètode *parserFile* amb els paràmetres actualitzats. El resultat es guarda a la variable *Assembly* resultant.

El mètode *createParts* crida directament la funció *makeCreation*. Al resultat que s'obtingui del tipus Part se li assignarà informació addicional, tal com el nom i el *path*, i s'inclourà a la variable *Assembly* resultat.

El mètode *createOperations* simplement crida el mètode *makeOperation*.

L'últim mètode *applyMotorFunction* comprova si el paràmetre d'entrada és cert, i si és així, es crida el mètode *assigMotor*.

9.2.4.3 - Procés *Execute File*: creació d'elements

El mètode que crida a la classe *CreatorService* per generar nous elements és el *makeCreation*, inclós dintre la classe *Parser*.

```
private Part makeCreation(Object operation) {
     Part result = new Part();
     //Check correct type Assignation
     if (operation.GetType() == typeof(Assignation)) {
          Ioperation op = (Ioperation)operation;
          string op_name = op.opname.Trim();

          //Check type to create
          if (op_name == "new gear") {
               NewGear(result, op);
          }
          else if (op_name == "new axis") {
               NewShaft(result, op);
          }
          else if (op_name == "new bevel") {
               NewBevel(result, op);
          }
          else {
               System.Console.WriteLine("Object creation type " + op_name +
                                                                  " incorrect.");
          }
     }
     else {
          System.Console.WriteLine("Object creation type " + operation.GetType() +
                                                                  " incorrect.");
     }
     return result;
}
```

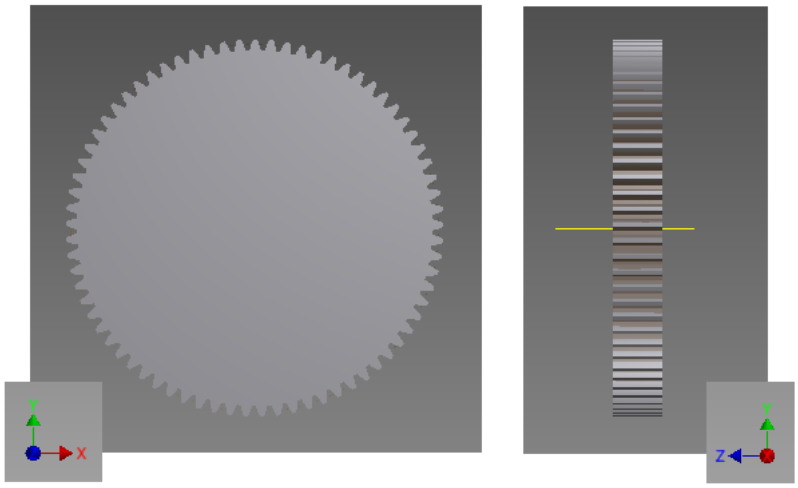
La principal funció d'aquest mètode és identificar el tipus d'element a crear.

A continuació podem veure el procés que segueix cada una de les opcions per crear les peces desitjades i els resultats esperats. Tots els mètodes de creació tenen en comú el

procés de parsejar els paràmetres propis de cada mòdul corresponent del fitxer XML i assignar les propietats específiques de cada tipus a la variable *Part result*.

```
private void NewGear(Part result, Ioperation op) {
     double diameter = Double.Parse(op.parameters[0],
                            System.Globalization.NumberStyles.AllowDecimalPoint,
                           System.Globalization.NumberFormatInfo.InvariantInfo);
     result.type = "gear";
     result.element = creator.createGear(diameter);
     result.eixAux = creator.createAxisAux();

     //Join gear with new axis aux
     constrainer.constraintAxis(result.element, result.eixAux, 3, 4, 0);
}
```

Inicialment es parseja el paràmetre indicat en el mòdul del fitxer XML per obtenir el valor del diàmetre de l'engranatge cilíndric. Amb el valor obtingut es crida la funció *createGear*. Adicionalment, també es crea un eix auxiliar el qual anirà unit amb l'eix central de l'engranatge. La funció encarregada d'aplicar la restricció entre els eixos és la *constraintAxis*, propia de la classe *ConstraintService*.



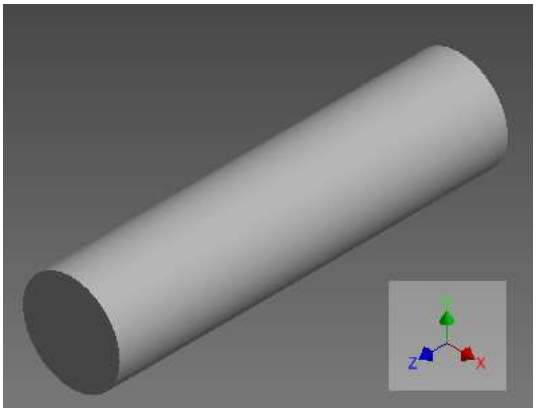
9.2.4.3.1 - Creació d'un engranatge cilíndric amb eix auxiliar

Per construir un eix de transmissió es segueixen els següents passos:

```
private void NewShaft(Part result, Ioperation op) {
     double diameter = Double.Parse(op.parameters[0],
                                System.Globalization.NumberStyles.AllowDecimalPoint,
                               System.Globalization.NumberFormatInfo.InvariantInfo);
     double longitud = Double.Parse(op.parameters[1],
                                System.Globalization.NumberStyles.AllowDecimalPoint,
                               System.Globalization.NumberFormatInfo.InvariantInfo);
     result.type = "axis";
     result.element = creator.createShaft(diameter, longitud);
}
```

Com tots els nous elements, inicialment s'obtenen els paràmetres necessaris per crear un nou eix de transmissió, en aquest cas el diàmetre i la longitud de l'eix. Amb els

valors aconseguits es fa una crida al *createShaft(diàmetre, longitud)*. El resultat, que consisteix en un nou eix de transmissió, s'assigna a les propietats del *Part result.*

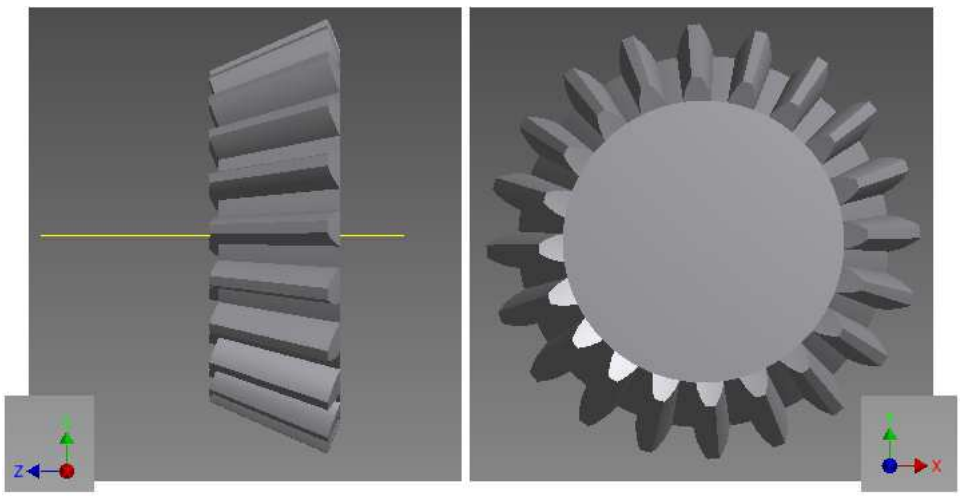


9.2.4.3.2 - Eix de transmissió

Finalment, per construir un engranatge cònic és necessari seguir els següents passos:

```
private void NewBevel(Part result, Ioperation op) {
     int teethG1 = int.Parse(op.parameters[0],
                                 System.Globalization.NumberStyles.AllowDecimalPoint,
                                System.Globalization.NumberFormatInfo.InvariantInfo);
     result.type = "bevel";
     result.element = creator.createBevel(teethG1);
     result.eixAux = creator.createAxisAux();

     //Join occ1 with new axis aux
     constrainer.constraintAxis(result.element, result.eixAux, 3, 4, 0);
}
```

En aquest cas, per construir un engranatge cònic només necessitem saber el número de dents que tindrà. Un cop parsejat el paràmetre, fem la crida al *createBevel*. El resultat s'assignarà a la variable *Part result*. Addicionalment, igual com passa amb els engranatges cilíndrics, necessitem crear un eix auxiliar, que anirà restringit coincident amb l'eix central de l'engranatge obtingut.



9.2.4.3.3 - Engranatge cònic amb eix auxiliar

9.2.4.4 - Procés *Execute File*: crear engranatges cilíndrics, cònics i eixos

Com s'ha comentat anteriorment, la funció *makeCreator* cridarà a un dels mètodes de creació propis de la classe *CreatorService.* Aquests mètodes són: *createGear, createShaft, createBevel* i *createAxisAux*.

Encara que no s'utilitza en el procés de creació, existeix un mètode més, el *createPlaneAux,* inclós en el procés d'aplicació d'operacions, que es pot incloure en aquest mateix grup.

El projecte disposa d'uns fitxers, amb extensió .ipt, on contenen els elements per defecte que l'aplicació coneix per treballar-hi. Aquests fitxers s'anomenen *templates*. Tots els mètodes de creació segueixen un mateix patró per a realitzar una còpia dels *templates* i modificar els paràmetres per defecte amb els establerts per l'usuari. El procés detallat és el següent:

1. Generar l'adreça *file* destí del *template*
2. Si ja existeix un fitxer *file,* eliminar-lo
3. Copiar el fitxer base *TEMPLATE* al fitxer *file*.
4. Obtenir l'assemblatge actiu d'Inventor
5. Generar matriu d'Inventor
6. Obrir fitxer còpia a l'assemblatge actual
7. Obtenir paràmetres i fòrmules del model còpia
8. Modificar paràmetres
9. Actualitzar i retornar resultat

Basant-nos amb el codi de creació per a engranatges cilíndrics, podem veure, resaltat en groc, les parts que cal adaptar per a cada cas:

```
public ComponentOccurrence createGear(double diameter) {
     //Genering template copy
     String file = GEAR_TEMP_FILE + this.num_gears + ".ipt";
     try {
          if (System.IO.File.Exists(file)) {
               System.IO.File.Delete(file);
          }
          System.IO.File.Copy(GEAR_TEMPLATE, file);
          num_gears++;
     }
     catch (Exception) {
          MessageBox.Show("Gear File template not found.");
     }

     // Get active assembly document
     AssemblyDocument asmDoc = (AssemblyDocument)_invApp.ActiveDocument;

     // Create the ComponentOccurrence and add Part
     Inventor.Matrix positionMatrix = _invApp.TransientGeometry.CreateMatrix();
     ComponentOccurrence result =
          asmDoc.ComponentDefinition.Occurrences.Add(file, positionMatrix);
```

```
        result.Grounded = false;

        //Configure Part
        PartDocument doc = (PartDocument)result.Definition.Document;
        PartComponentDefinition compDef = doc.ComponentDefinition;
        ModelParameters mParam = compDef.Parameters.ModelParameters;

        double teeth = (diameter-0.4) * FACTOR; //Calcular el numero de dents
        mParam["da_da"].Value = diameter; //Diametre extern
        mParam["da_z"].Value = Math.Round(teeth); //Num. Dents

        asmDoc.Update();

        return result;
}
```

Pels mètodes *createGear, createShaft* i *createBevel,* l'objectiu del procés anterior és la part final, on es modifiquen els paràmetres del model copiat, basat en el *TEMPLATE* desitjat. Per veure les fòrmules i els valors de les variables dels *templates* utilitzats en el projecte, veure l'annex 14.3.

Al crear un engranatge cònic, s'assigna el paràmetre d'entrada, el número de dents, al paràmetre "da_z".

```
mParam["da_z"].Value = teeth; //Num. Dents Propi
```

Al crear un eix de transmissió, s'assignen els paràmetres d'entrada, el diàmetre i la longitud, als paràmetres "d0" i "d1", en el mateix ordre.

```
mParam["d0"].Value = diameter;  //Diameter
mParam["d1"].Value = longitud;  //Length
```

Els fitxers que contenen els models dels objectes tenen un seguit de fòrmules i variables que, a l'especificar els paràmetres anteriors, modifiquen els valors resultants de les fòrmules i re-dimensionen la peça continguda.

Pels mètodes *createAxisAux* i *createPlaneAux*, es suprimeix el procés de modificar els paràmetres i s'afageix el procés de guardar el resultat a la llista corresponent (*axisAuxList, planeAuxList*).

9.2.4.5 - Problemes i solucions: crear engranatges cilíndrics i cònics

Al llarg del procés de desenvolupament del projecte, els mètodes disenyats per crear els elements no han variat gaire. No obstant, per poder fer possible els canvis que calia fer als mètodes de restriccions, s'ha hagut d'afegir la creació d'un eix auxiliar, paral·lel i coincident amb el propi eix dels engranatges. Aquesta pràctica ha fet possible la col·locació dels elements a qualsevol lloc de l'espai i poder fixar-los a la posició on es trobin en el moment d'activar el motor, aconseguint que no es moguin de posició, però sí puguin rotar.

Cal destacar que, al final del procés de creació, tindrem un fitxer *.ipt* per cada element que s'hagi creat: engranatges, eixos auxiliars,... guardats a la carpeta */temp* del directori arrel. Quan es carrega un fitxer *.ipt,* amb els elements per defecte (guardats al directori */templates*), *Autodesk Inventor* manté un valor de referència sobre el fitxer, i no sobre l'element lògic carregat, fent que, si carreguem diverses vegades un fitxer i modifiquem els paràmetres només d'un dels elements carregats, el resultat serà que tots els elements que partien del mateix fitxer es modifiquin per igual. Per aquesta raó, es guarda una còpia del fitxer per defecte per cada objecte que es crea.

9.2.4.6 - Procés *Execute File*: aplicació d'operacions

El mètode de la classe *Parser* encarregat de distingir el tipus d'operació a aplicar entre els elements indicats és el *makeOperation*.

```
private void makeOperation(Object operation, Assembly assem) {
     if (operation.GetType() == typeof(Operation)) {
          Ioperation op = (Ioperation)operation;
          string op_name = op.opname.Trim();
          if (op_name == "join") {
               Part obj1 = assem.getPart((string)op.parameters[0]);
               Part obj2 = assem.getPart((string)op.parameters[1]);

               if (obj1.type == "gear" && obj2.type == "gear") {
                    gearWithGear(op, obj1, obj2);
               }
               else if ((obj1.type == "axis" && obj2.type == "gear")
                                   || (obj1.type == "gear" && obj2.type == "axis")) {
                    gearWithShaft(op, obj1, obj2);
               }
               else if (obj1.type == "bevel" && obj2.type == "bevel") {
                    bevelWithBevel(op, obj1, obj2);
               }
               else if ((obj1.type == "axis" && obj2.type == "bevel")
                                  || (obj1.type == "bevel" && obj2.type == "axis")) {
                    bevelWithShaft(obj1, obj2);
               }
          }
     }
}
```

Un cop indicat el tipus d'unió que es vol realitzar, es criden els mètodes privats per a realitzar les operacions de cada cas.

9.2.4.7 - Procés *Execute File*: mètodes de restricció

Abans de començar a explicar els procéssos d'unió entre elements cal saber i entendre quin patró segueixen els mètodes de restricció i la diferència entre tots els mètodes per poder personalitzar les restriccions a aplicar.

Tot els mètodes, que es troben a la clase *ConstraintService*, segueixen un mateix patró. Tot i així no s'han unificat en un mètode general degut a les diferències de restriccions

pròpies de l'*Autodesk Inventor* i dels elements a restringir: plans, eixos, punts, superfícies,... .

L'escritura general dels mètodes està formada pels *ComponentOccurrences* a unir, els elements del sistema de coordenades de l'origen o pròpis dels elements (punts, eixos, plans, superfícies, ...) que es volen utilitzar i, en alguns casos, altra informació addicional com angles o distàncies.

L'esquema general és:

1. Obtenir el document de l'assemblatge actiu (*AssemblyDocument*) i la seva definició (*AssemblyComponentDefinition*).
2. Obtenir les definicions dels *ComponentOccurrences* passats com a paràmetres (*PartComponentDefinition*).
3. Obtenir els elements de treball (punts, eixos, plans,...) a partir de les definicions:
    - En el cas dels elements dels objectes, a partir de la definició obtinguda en el punt anterior (tipus *PartComponentDefinition*).
    - En el cas d'elements del sistema de coordenades de l'assemblatge, a partir de la definició (tipus *AssemblyComponentDefinition*).
4. Obtenir els *Proxy*s dels elements de treball.
5. Aplicar les restriccions sobre l'*AssemblyComponentDefinition* passant com a paràmetres els *Proxy*s i valors adicionals.
6. Actualitzar l'assemblatge per visualitzar els canvis.

El mètode només obtindrà les referències de peces creades per nosaltres. Els elements pròpis del sistema de referència s'obtenen directament de l'assemblatge

Tots els elements que componen l'espai de l'assemblatge i les peces (punts, plans,...) es troben inclosos en *arrays* dintre la informació pròpia de cada definició, sempre en la mateixa posició. Per aquesta raó, els mètodes esperen rebre, per paràmetre, valors enters coneguts per referir-se als índexos dels element a restringir. L'ordre dels índexs correspon a l'ordre del pas per paràmetres dels *ComponentOccurrences* i, després, als elements de l'origen.

A continuació veurem un exemple. El següent mètode uneix el pla indicat per l'index "index" de l'element "occ1" amb el pla de l'origen indicat per l'index "indexOrigin".

```
private void constraintPlaneOriginPlane(ComponentOccurrence occ1, int index, int
                                                                    indexOrigin) {
     // Get active assembly document and Component Definition
     AssemblyDocument asmDoc = (AssemblyDocument)_invApp.ActiveDocument;
     AssemblyComponentDefinition aCompDef = asmDoc.ComponentDefinition;

     // Get the Definition from occurrence
     PartComponentDefinition pPartDef1 = (PartComponentDefinition)occ1.Definition;
```

```
        // Get work elements from the parts
        WorkPlane pPartPlane1 = pPartDef1.WorkPlanes[index];
        WorkPlane pPartPlane3 = aCompDef.WorkPlanes[indexOrigin];

        // Convert the work elements to assembly geometry proxy
        object planeProxy1 = null;
        occ1.CreateGeometryProxy(pPartPlane1, out planeProxy1);
        WorkPlaneProxy pWorkPlaneProxy1 = (WorkPlaneProxy)planeProxy1;

        // Assemble parts
        aCompDef.Constraints.AddFlushConstraint(pWorkPlaneProxy1, pPartPlane3, 0.0);

        asmDoc.Update();
}
```

Si en lloc de restringir un element d'una peça amb l'origen, es fes amb una altra peça, el tracte seria equivalent al primer element, substituint les part que tracten l'element de l'origen pel nou.

Les parts senyalades en groc del codi anterior indiquen els tipus dels elements a adaptar segons es vulgui. La següent taula ofereix les diverses opcions utilitzades en el projecte.

Elements de treball		
Tipus a obtenir	Element que el conté	Comanda
WorkPoint	Peça	partDefinition.WorkPoints[index];
	Origen	-
WorkAxis	Peça	partDefinition.WorkAxes[index];
	Origen	-
WorkPlane	Peça	partDefinition.WorkPlanes[index];
	Origen	assemblyDefinition.WorkPlanes[index];
Face	Peça	partDefinition.WorkSurfaces[1].SurfaceBodies[1].Faces[1];
		partDefinition.Features["Fijar cuerpo"].Faces[index];
	Origen	-

Elements Proxy
Tipus a obtenir
WorkPointProxy
WorkAxisProxy
WorkPlaneProxy
FaceProxy

Els mètodes pròpis d'*Autodesk Inventor* per restringir utilitzats en el projecte i els paràmetres que se li passen són:

- AddFlushConstraint: requereix el valor de la distància entre els elements restringits (zero per defecte).
- AddMateConstraint: requereix el valor de la distància entre els elements restringits (zero per defecte).
- AddTangentConstraint: requereix el valor de la distància entre els elements restringits (zero per defecte).
- AddAngleConstraint: requereix el valor en graus de l'angle entre els elements.

- AddRotateRotateConstraint: requereix el valor de transmissió (divisió dels diàmetres dels engranatges) i un valor booleà per determinar si giren en el mateix sentit o no.

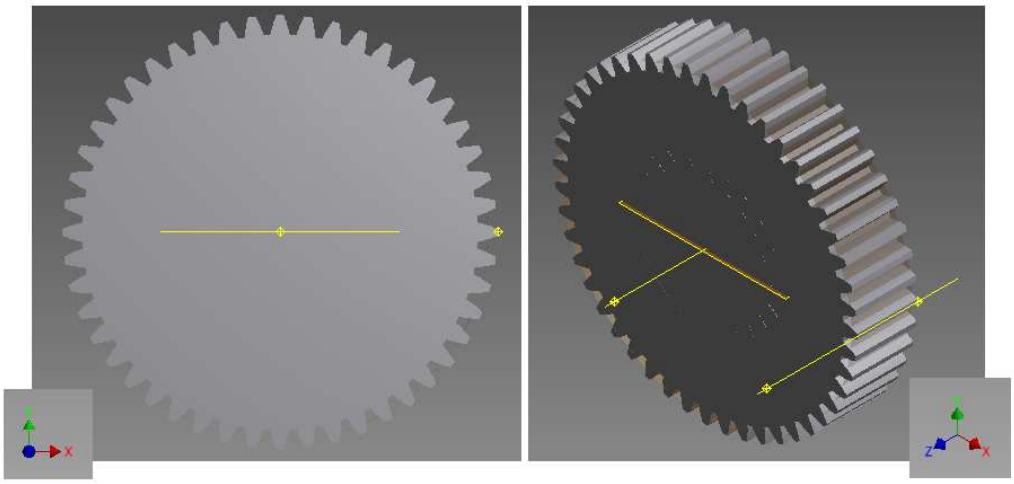
9.2.4.8 - Procés *Execute File*: unió entre engranatges cilíndrics

La funcionalitat principal del projecte és poder unir dos engranatges entre si, de tal manera que, al moure'n un, el moviment es transmeti a l'altre. De per si, aquest comportament es pot aplicar amb una única restricció, però, si no s'afageixen altres restriccions, l'assemblatge resultant no s'assemblarà en res a un mecanisme que es pugui construir a la realitat. És a dir, tot i que dintre del simulador els engranatges giraran en sentit oposat, transmatent-se el moviment, es poden trobar a prou distància, o fins i tot en plans diferents, causant que visualment ni hi hagi contacte entre ells.

Per aquesta raó, el mètode *gearWithGear*, igual com la resta d'operacions d'unió, apliquen diverses operacions per aconseguir un resultat el màxim real possible.

```
private void gearWithGear(Ioperation op, Part obj1, Part obj2) {
     ComponentOccurrence gear1 = obj1.element;
     ComponentOccurrence gear2 = obj2.element;
     ComponentOccurrence plane = creator.createPlaneAux();

     double angleComplert = double.Parse(op.parameters[2],
                                 System.Globalization.NumberStyles.AllowDecimalPoint,
                                System.Globalization.NumberFormatInfo.InvariantInfo);
     double angle = angleComplert % 360;
     obj1.addAngle(obj2.path, angle);

     //La dist es el diametre de gear1 o gear2 mes gran
     double dist = (getDiameterSpurGear(gear1) > getDiameterSpurGear(gear2) ?
                           getDiameterSpurGear(gear1) : getDiameterSpurGear(gear2));

     obj1.eixAux.Grounded = true;
     //Constraints fixes
     constrainer.constraintPlanes(gear2, gear1, 3, 3, "0");  //g1.XY amb g2.XY

     //Posicio inicial
     defineQuadrant(gear1, gear2, angle, dist);
     makeTranslation(gear1, plane, 0, 0);
     //Constraints fixes
     constrainer.constraintTangentSup(gear1, gear2); //g1.Sup tangent amb g2.Sup
     constrainer.constraintRotatePlanes(gear1, gear2, 3, 3, false);  //g1.XY gira en
                                                                      contra de g2.XY
     //Constraints de col·locacio
     constrainer.constraintAxis(gear1, plane, 3, 4, 0);  //g1.z amb plane.eix
     constrainer.constraintAnglePlaneOriginPlane(plane, 4, 2, angle);   //planeAux
                                                                  amb "angle" a o.XZ
     constrainer.constraintAxisPlane(gear2, plane, 3, 4);    //g2.z amb planeAux

     obj1.eixAux.Grounded = false;
}
```

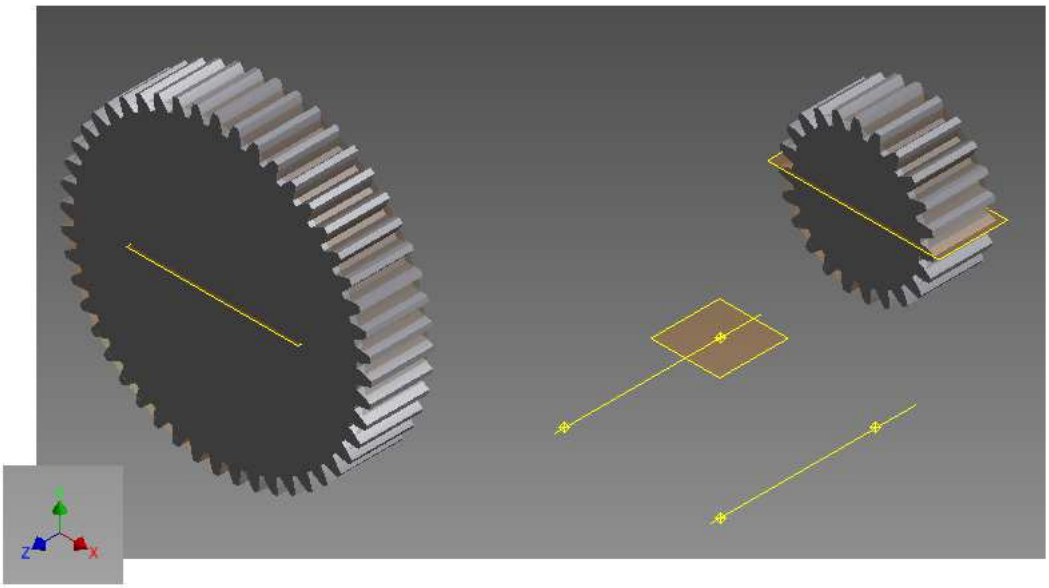
Inicialment, la funció obté els *ComponentOccurrences* dels dos engranatges a unir. A més, crida la funció per crear un pla auxiliar (veure procès anterior de creació). Aquest pla servirà per fixar el segon engranatge respecte el primer en un angle determinat.



9.2.4.8.1 - Pas 1: elements creats

A la imatge 9.2.4.8.1 s'ha respectat la posició inicial dels elements quan són carregats a la plataforma. Podem veure com els dos engranatges i el pla auxiliar estan solapats. Per defecte, es carreguen tots sobre el punt d'origen de l'espai de coordenades.

El pla està creat sobre dos eixos auxiliars que, en el seu torn, estan creats sobre dos punts auxiliars. És per aquesta raó que veiem eixos de color groc a les imatges coincidents amb el pla auxilar.



9.2.4.8.2 - Elements per separat

Per poder veure millor l'aplicació de les restriccions, el posicionament de les peces es mantindrà igual que a la figura 9.2.4.8.2.

Un cop carregat els elements, s'obtenen els valors dels paràmetres necessàris. Primer s'obté el valor en graus de l'angle especificat en el fitxer XML. Després s'obté el valor del diàmetre més gran dels dos engranatges.

Després d'obtenr els valors anteriors, comença el procés d'unió.

Primer es fixa l'eix auxiliar del primer engranatge.

```
obj1.eixAux.Grounded = true;
```

L'objecte de tipus *Part* conté la referència de l'eix auxiliar. Cal recordar que, en el procés de creació d'un engranatge es crea un eix auxiliar, unit de manera paral·lela i coincident amb el propi eix central de l'engrenatge.

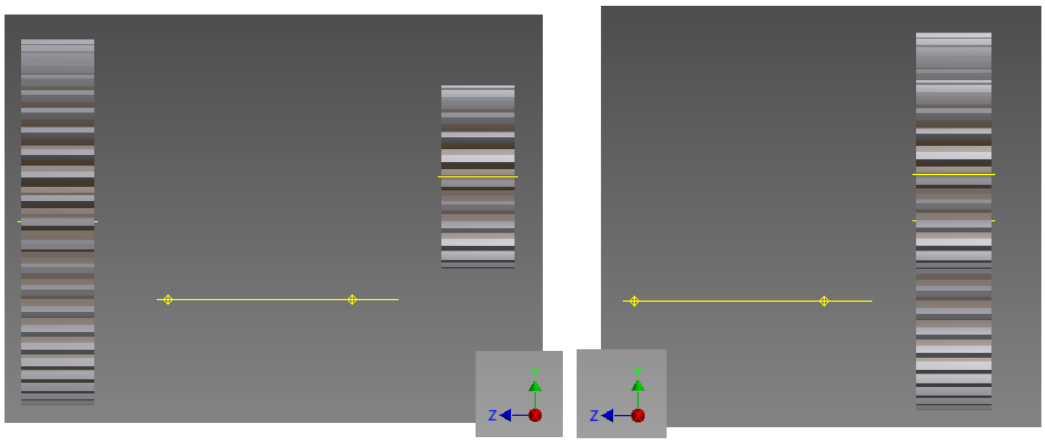
D'aquesta manera ens assegurem que el primer engranatge no es mourà i que totes les restriccions entre ambdós engranatges s'aplicaran sempre respecte l'engranatge fixat i de la manera desitjada.

De l'exemple, és l'engranatge petit el què es quedarà fixat.

Tot seguit es comencen a aplicar les restriccions d'unió i posicionament.

```
constrainer.constraintPlanes(gear2, gear1, 3, 3, "0");
```

Primer es col·loquen els dos engranatges en el mateix pla. El pla està indicat per l'índex 3 i correspont al pla de les XY.

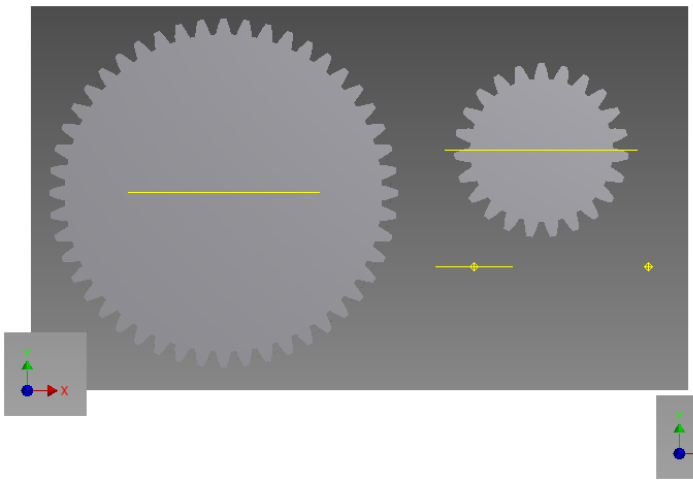


9.2.4.8.3a - Abans de posicionar al mateix pla

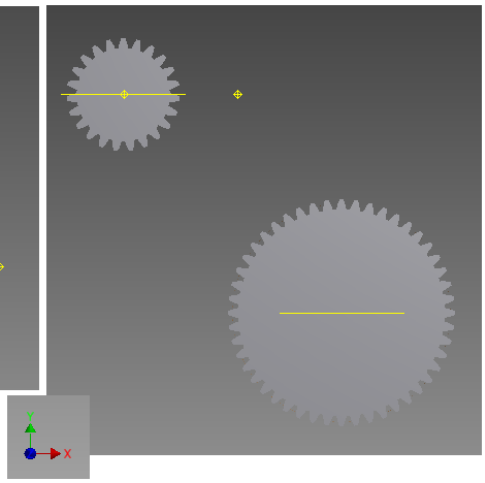
9.2.4.8.3b - Després de posicionar al mateix pla

Amb l'angle i la distància obtinguda del diàmetre més gran, posicionem el segon engranatge en el quadrant respecte el primer que s'aproximi a l'angle establert. En aquest cas, i sabent que l'ordre dels angles va de l'eix de les X i en sentit de les agulles del rellotge, com que s'ha de posicionar a un angle de 30 graus, el segon engranatge es posarà al quadrant de baix a la dreta (veure imatges 9.2.4.8.3a i b).

Aprofitem aquest pas per posicionar el pla auxiliar al centre de l'engranatge fixat 1.



9.2.4.8.4a - Abans de posicionar el *gear2* i el pla auxiliar

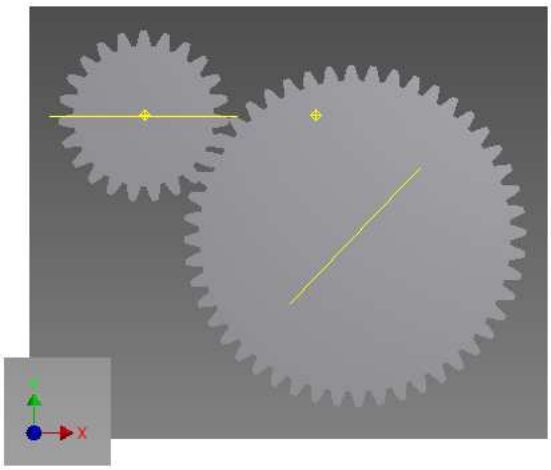


9.2.4.8.4b - Després de posicionar el *gear2* i el pla auxiliar

Els següents processos serveixen per unir els engranatges i aplicar la dependència de moviment.

```
constrainer.constraintTangentSup(gear1, gear2);
constrainer.constraintRotatePlanes(gear1, gear2, 3, 3, false);
```

D'aquesta manera visualment els engranatges es tocaran i, al moure'n un, l'altre també girarà.



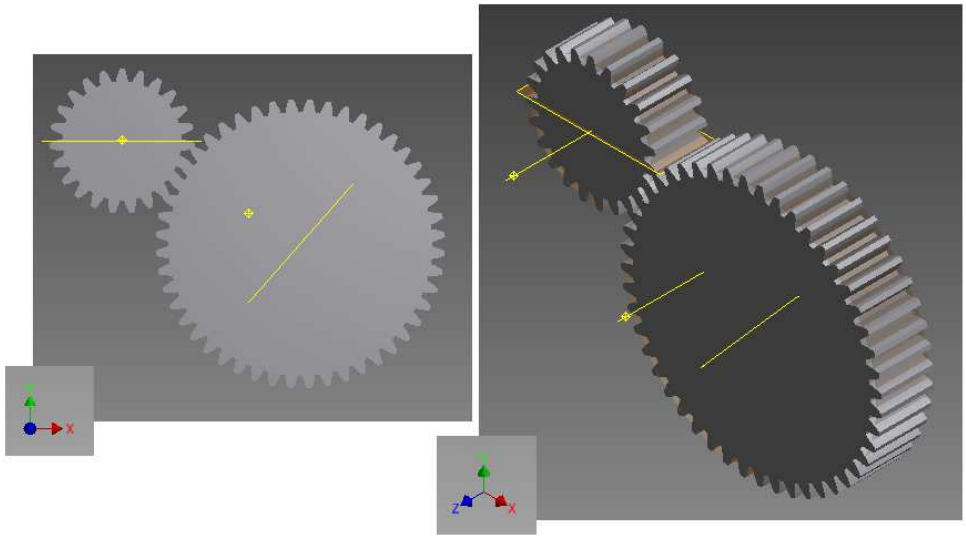
9.2.4.8.5 - Unió del engranatges

Com podem veure a la imatge 9.2.4.8.5, el segon engranatge s'ha girat (cal fixar-se en el seu pla). I les dents no acaben de coincidir. Pel motiu que el solapament de les dents és un problema visual, però no mecànic, en aquesta versió del projecte això no es resol.

En aquest punt cal posicionar el segon engranatge a 30 graus del primer i assegurar-se que no es mogui. Per fer-ho utilitzarem el pla auxiliar i les comandes següents:

```
constrainer.constraintAxis(gear1, plane, 3, 4, 0);
constrainer.constraintAnglePlaneOriginPlane(plane, 4, 2, angle);
constrainer.constraintAxisPlane(gear2, plane, 3, 4);
```

El resultat final és:



9.2.4.8.6 - Resultat final d'unió de dos engranatges

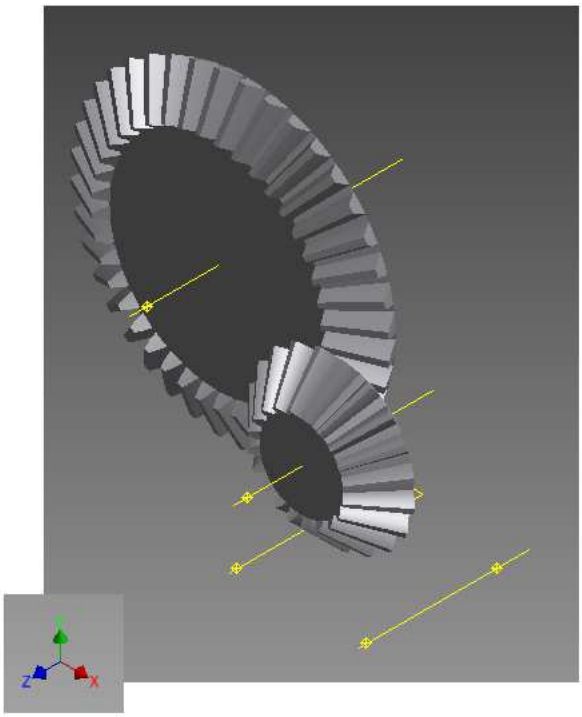
9.2.4.9 - Procés *Execute File*: unió entre engranatges cònics

Els engranatges cònics funcionen de manera similar que els engranatges cilíndrics, però sense treballar en el mateix pla. El mètode que fa possible la unió entre engranatges cònics és el *bevelWithBevel*.

```
private void bevelWithBevel(Ioperation op, Part obj1, Part obj2) {
     ComponentOccurrence gear1 = obj1.element;
     ComponentOccurrence gear2 = obj2.element;
     ComponentOccurrence plane = creator.createPlaneAux();

     double angleComplert = double.Parse(op.parameters[2],
                                 System.Globalization.NumberStyles.AllowDecimalPoint,
                                System.Globalization.NumberFormatInfo.InvariantInfo);
     double angle = angleComplert % 360;
     obj2.angle = angle;
     obj1.addAngle(obj2.path, angle);

     //La dist es el diametre de gear1 o gear2 mes gran
     double dist = (getDiameterBevel(gear1) > getDiameterBevel(gear2) ?
                                  getDiameterBevel(gear1) : getDiameterBevel(gear2));

     relateTeeth(gear1, gear2);

     //Posicio inicial
     makeTranslation(gear1, plane, 0, 0); //Moure pla

     obj1.eixAux.Grounded = true;
     //Constraints fixes
     constrainer.constraintPoints(gear1, gear2, 1, 1);   //g1.p1 amb g2.p1
     constrainer.constraintAngleAxis(gear1, gear2, 3, 3, 90);    //g1.z amb g2.z a 90
                                                                        graus
     constrainer.constraintRotateAxis(gear1, gear2, 3, 3, false);    //g1.z gira amb
```

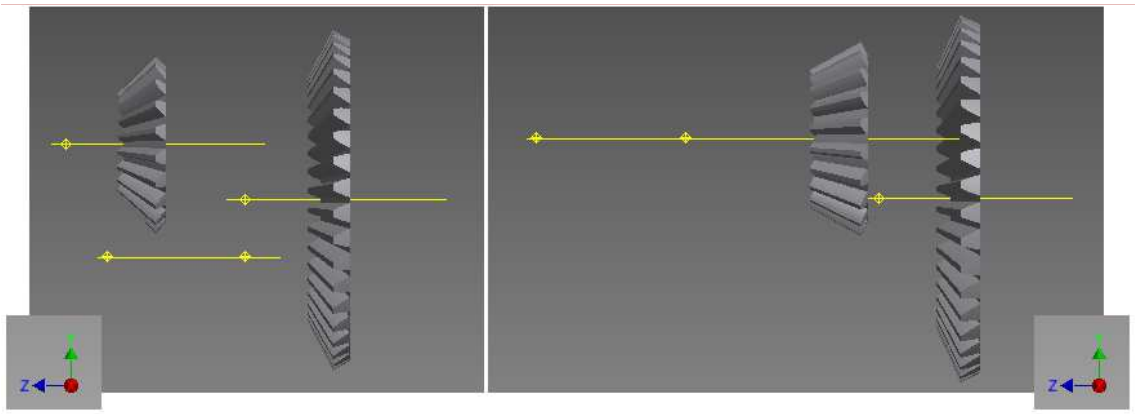
```
                                                          g2.z sentit contrari

    makeRotation(gear2, (angle - 90) * Math.PI / 180, 0, 1, 0);

    //Constraints de col·locacio
    constrainer.constraintAxis(gear1, plane, 3, 4, 0);  //g1.z amb plane.eix
    constrainer.constraintAxisPlane(gear2, plane, 3, 4);    //g2.z amb planeAux
    constrainer.constraintAnglePlaneOriginPlane(plane, 4, 2, angle);   //planeAux
                                                              amb "angle" a o.XZ

    obj1.eixAux.Grounded = false;
}
```

Com en el mètode d'unió d'engranatges cilíndrics, inicialment s'han carregat els dos elements a unir i es crea un pla auxiliar per fer possible la col·locació en un angle específic.



9.2.4.9.1 - Elements inicials per unió entre cònics

Posteriorment s'agafen els valors de les variables que es necessitaran. Un dels processos que s'aplica en aquest punt és el procés de relacionar les dents dels dos engranatges. És necessari definir dintre de les operacions pròpies dels engranatges cònics el número de dents que tindrà el component amb qui anirà unit, degut que hi ha operacions que utilitzen aquest valor per dimensionar correctament l'estructura. De per sí, a la hora de la creació només sabem el número de dents que volem a l'engranatge a crear. No sabrem fins arribat a les operacions d'unió amb quin engranatge s'unirà.



9.2.4.9.2 - L'abans (esquerra) i el després (dreta) de redimensionar

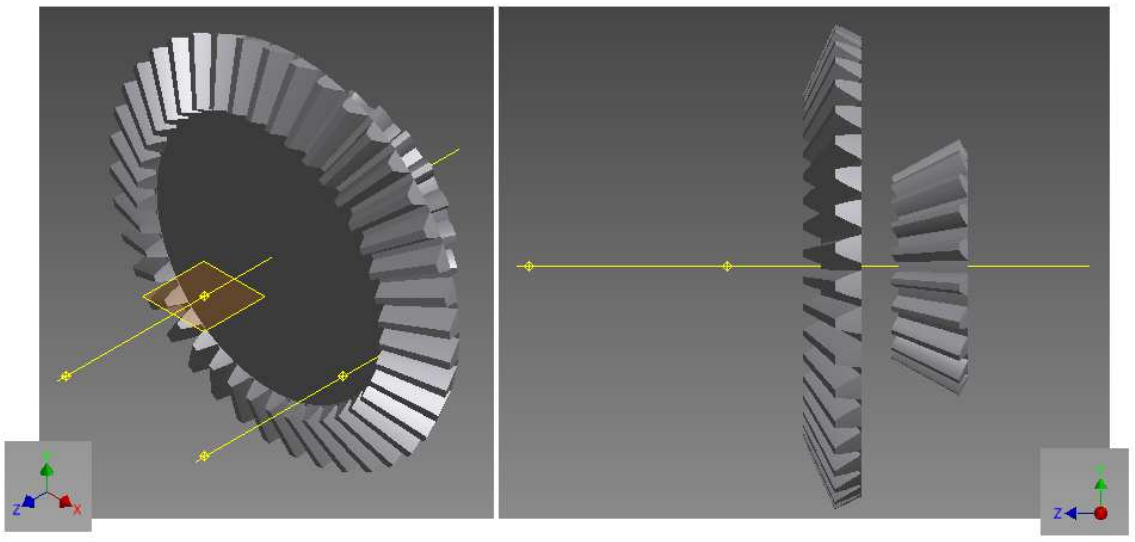
A la figura 9.2.4.9.2 veiem que a la imatge de la dreta el punt central de l'engranatge petit s'ha allunyat, fent que la peça sigui més gruixuda, i que la base de l'engranatge gran ha augmentat.

Aprofitem a la imatge 9.2.4.9.2 que ja tenim els engranatges cònics ben dimensionats per moure el pla auxiliar al centre del primer engranatge, i fixar l'eix auxiliar.

El bloc següent serveix per col·locar el segon engranatge al pla correcte, a 90 graus del primer, i aplicar la restricció de transmissió de moviment.

```
constrainer.constraintPoints(gear1, gear2, 1, 1);
constrainer.constraintAngleAxis(gear1, gear2, 3, 3, 90);
constrainer.constraintRotateAxis(gear1, gear2, 3, 3, false);
```

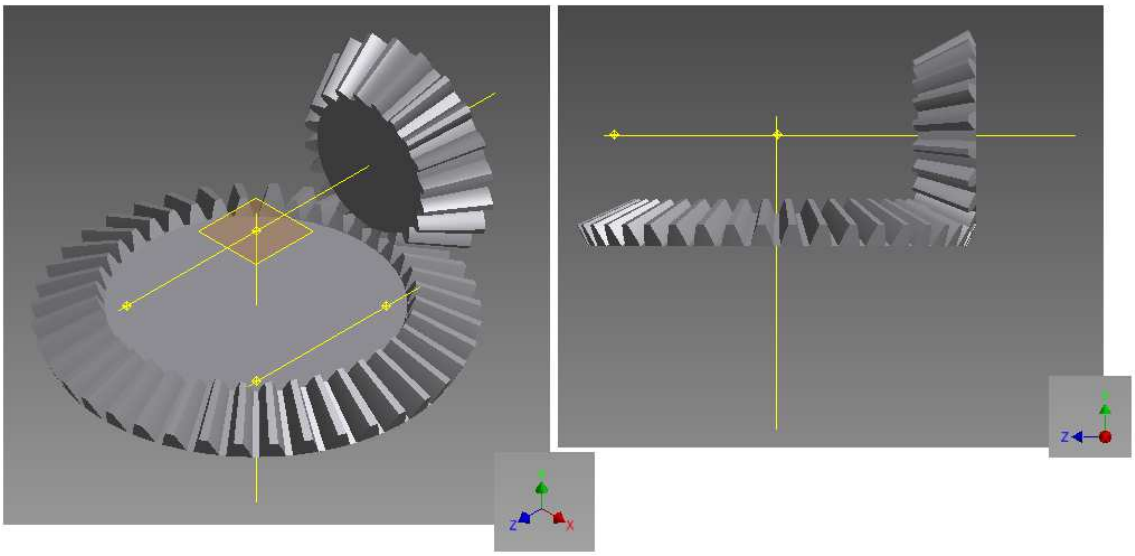
Els punts s'indiquen amb l'índex 1, i els eixos Z, que atravessen els engranatges pel centre, amb l'índex 3.



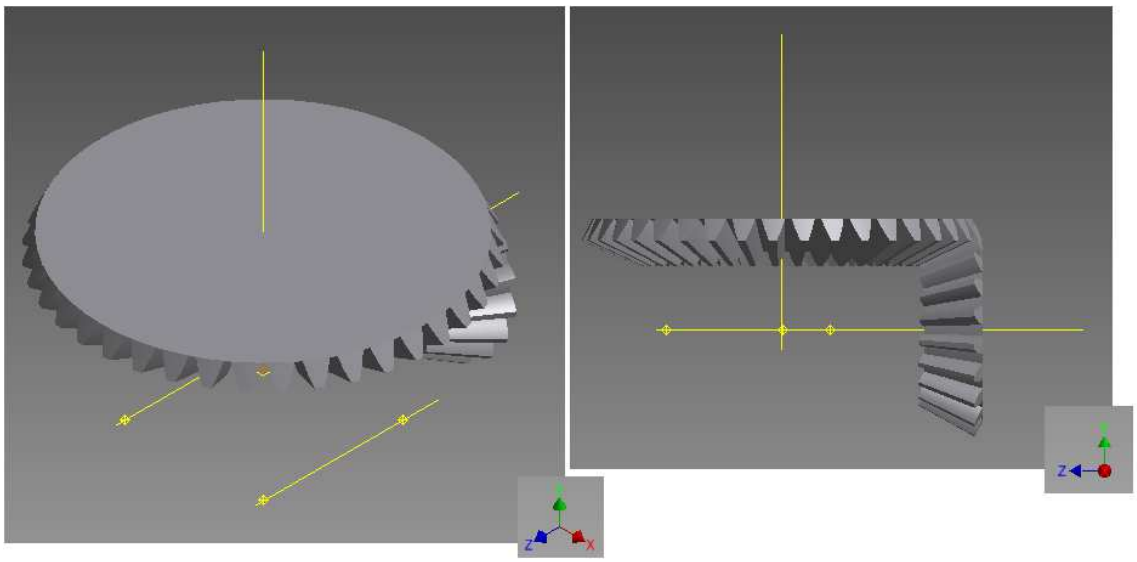
9.2.4.9.3 - Unió dels punts

El procés per posar el segon engranatge a 90 graus del primer és ambigu. És a dir, la posició en que es troba el segon engranatge fa que al girar-lo 90 graus de manera

vertical, compleixi la restricció tan si es mou cap amunt com cap a baix. Aquest factor depent de les dècimes amb què traballa internament l'*Inventor*, determinades per les posicions relatives dels elements i les seves dimensions. En tot cas, independentment de la orientació en aquest procés, el resultat final no ha de variar.



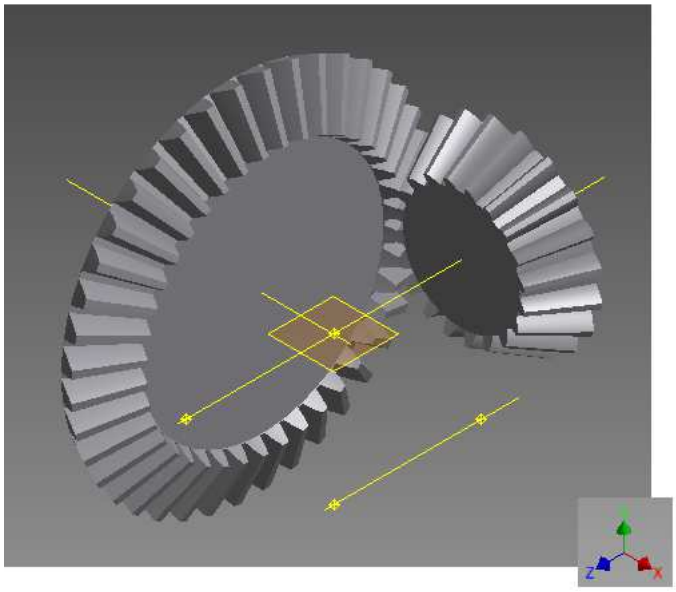
9.2.4.9.4a - Posició més comuna a l'aplicar el gir de 90 graus



9.2.4.9.4b - Posició poc frequent per l'aplicació del gir

La comanda de rotació posiciona el segon engranatge a l'espai més pròxim a l'angle desitjat.

```
makeRotation(gear2, (angle - 90) * Math.PI / 180, 0, 1, 0);
```

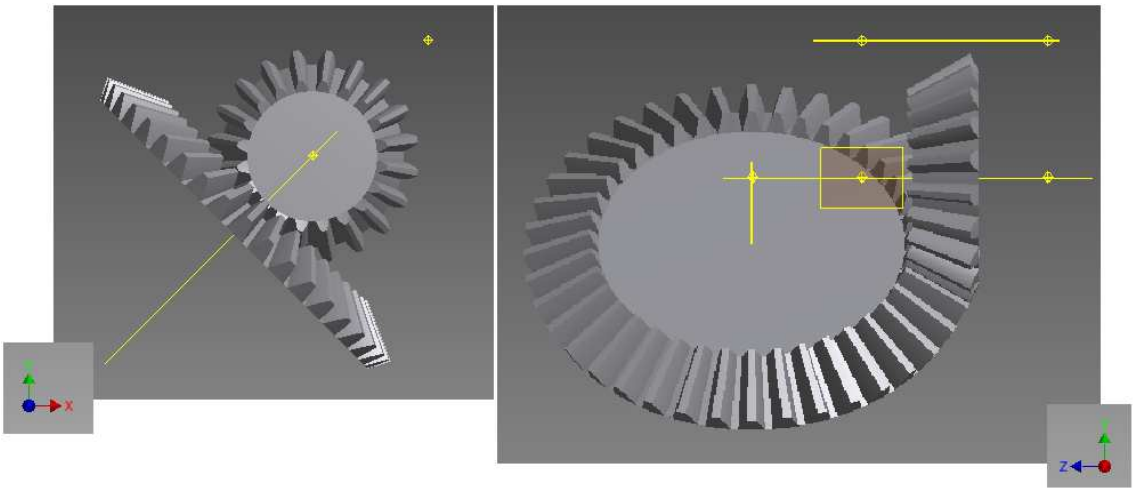


9.2.4.9.5 - Posició pròxima a l'angle final

Finalment el pla auxiliar es posiciona correctament a l'angle desitjat, per l'exemple 135 graus, i es fa coincidir amb el segon engranatge.

```
constrainer.constraintAxis(gear1, plane, 3, 4, 0);
constrainer.constraintAxisPlane(gear2, plane, 3, 4);
constrainer.constraintAnglePlaneOriginPlane(plane, 4, 2, angle);
```

El resultat final es pot veure a la figura 9.2.4.9.6.



9.2.4.9.6 - Resultat final d'unió d'engranatges cònics

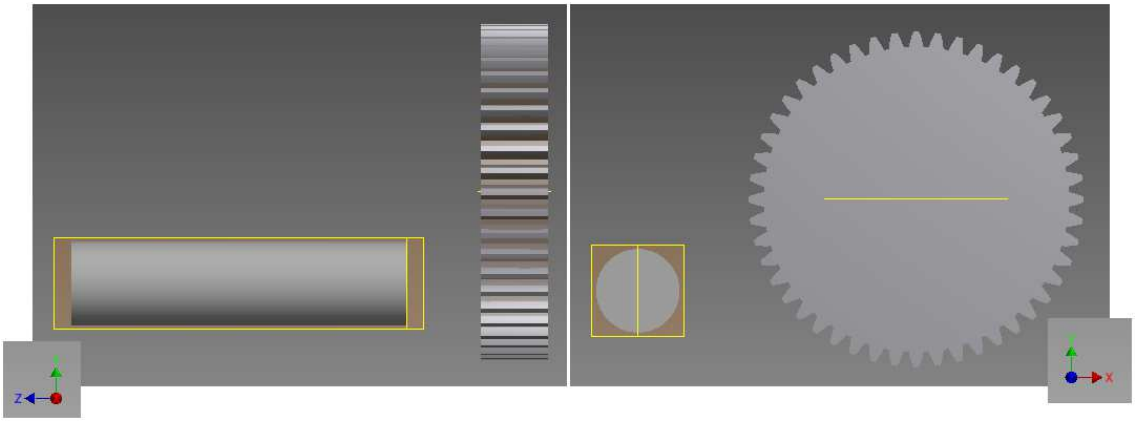
9.2.4.10 - Procés *Execute File*: Unió entre engranatge cilíndric i eix

Els processos d'unió amb eixos de transmissió no són tan complexos com els que uneixen engranatges entre si. Tot i així, el procés entre unir un eix amb un engranatge cònic o un engranatge cilíndric varia.

Per unir un eix amb un engranatge cilíndric cal executar el següent fragment de codi:

```
private void gearWithShaft(Ioperation op, Part obj1, Part obj2) {
     ComponentOccurrence gear = (obj1.type == "gear" ? obj1.element : obj2.element);
     ComponentOccurrence axis = (obj1.type == "axis" ? obj1.element : obj2.element);

     double param = double.Parse(op.parameters[2],
                                 System.Globalization.NumberStyles.AllowDecimalPoint,
                                System.Globalization.NumberFormatInfo.InvariantInfo);
     double dist = margeDist(axis, param);

     constrainer.constraintAxis(gear, axis, 3, 3, 0);
     constrainer.constraintPlanes(gear, axis, 3, 3, "-" + dist + "cm");
     constrainer.constraintPlanes(gear, axis, 1, 1, "0");
}
```

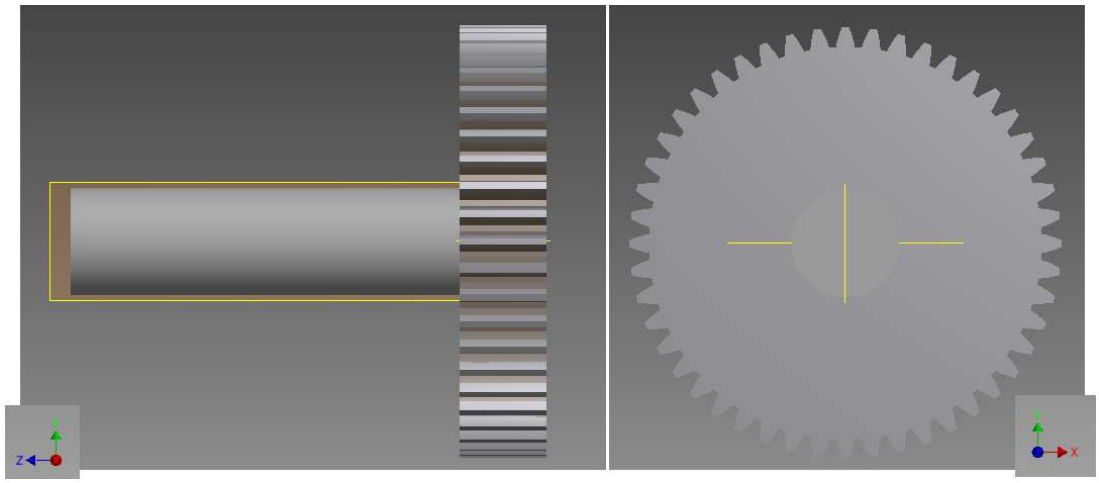
Inicialment carreguem les referències dels elements ja creats i obtenim el valor de la distància en què s'unirà l'eix amb l'engranatge. El mètode *margeDist* filtra aquest valor perquè no superi la longitud màxima que fa l'eix.



9.2.4.10.1 - Càrrega dels elements incials d'unió

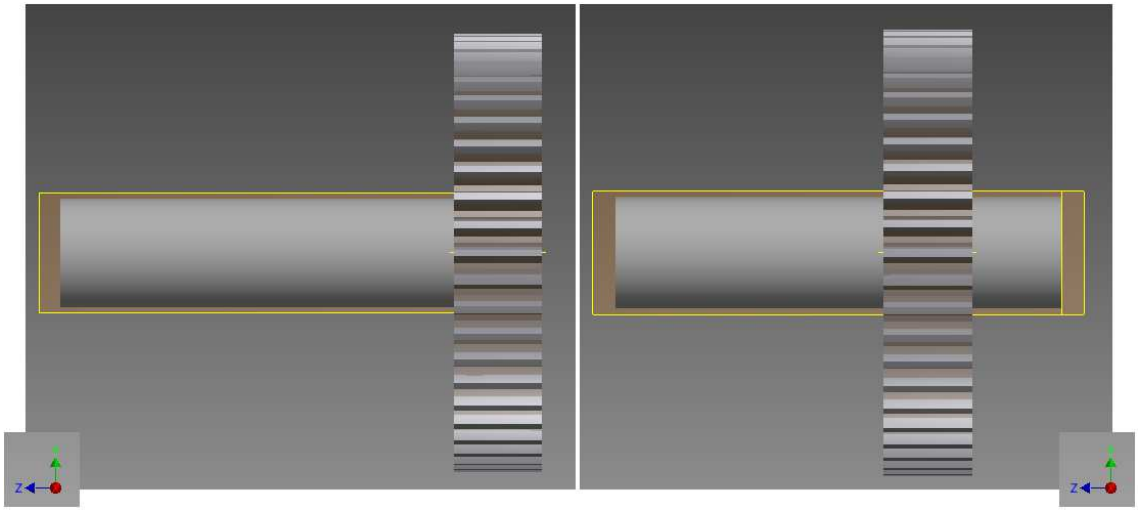
Per visualitzar més correctament l'orientació de l'eix de transmissió s'ha fet visible el pla de les YZ, que l'atravessa pel llarg, i el pla de les XY, que correspont a la seva base.

Tot seguit anivellem l'eix cèntric de l'engranatge amb l'equivalent de l'eix de transmissió, coincident amb el pla de les YZ.



9.2.4.10.2 - Anivellació dels eixos

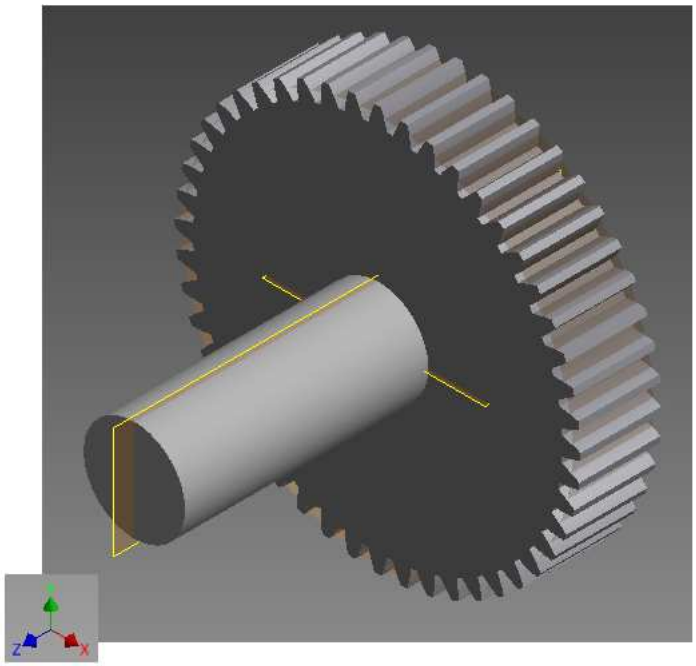
Finalment situem l'eix de transmissió, concretament el pla de la base XY, a la distància desitjada respecte la base de l'engranatge, en aquest cas a 2 centímetres.



9.2.4.10.3 - Abans i després de posicionar la distància de l'eix

En aquest últim pas s'ha aprofitat per fer coincident els plans de les YZ, indicats per l'índex 1. D'aquesta manera, al fer girar un engranatge, el pla coincident farà girar l'eix de transmissió.

El resultat final es pot veure a la imatge 9.2.4.10.4.



9.2.4.10.4 - Resultat final d'unió entre engranatge i eix

9.2.4.11 - Procés *Execute File*: unió entre engranatge cònic i eix

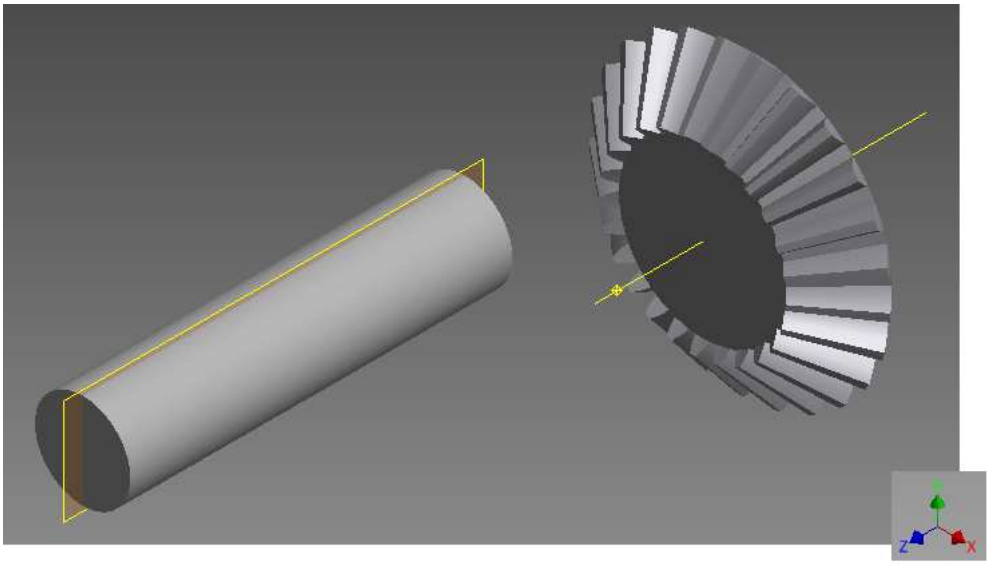
L'eix de transmissió s'uneix tan amb els engranatges cilíndrics com els cònics de manera molt similar. L'únic punt que varia és que, en lloc de fer coincidents el pla base

de l'eix amb el de l'engranatge (tingui o no una distància afegida), s'uneix amb la superfície de la cara més gran. A causa de que aquesta cara no coincideix amb cap dels plans de coordenades de l'engranatge cònic, s'ha hagut de fer d'aquesta manera.

```
private void bevelWithShaft(Ioperation op, Part obj1, Part obj2) {
     ComponentOccurrence bevel = (obj1.type == "bevel" ? obj1.element :
                                                                 obj2.element);
     ComponentOccurrence axis = (obj1.type == "axis" ? obj1.element : obj2.element);

     double param = double.Parse(op.parameters[2],
                                 System.Globalization.NumberStyles.AllowDecimalPoint,
                                System.Globalization.NumberFormatInfo.InvariantInfo);
     double dist = margeDist(axis, param);

     constrainer.constraintAxis(bevel, axis, 3, 3, 0);
     constrainer.constraintPlanes(bevel, axis, 1, 1, "0");
     constrainer.constraintFacesBevelAxis(bevel, axis, "-" + dist + "cm");
}
```

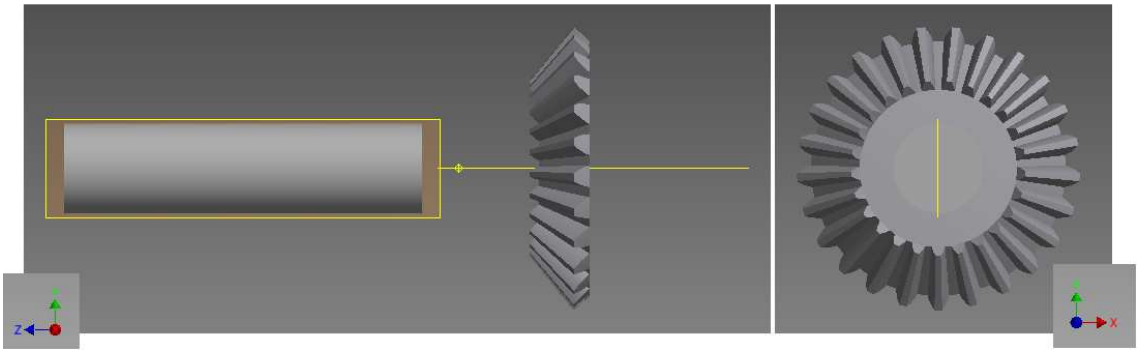
Visualment el procés és el mateix.

Inicialment es carreguen les peces i s'obté la distància d'unió entre l'engranatge i l'eix.



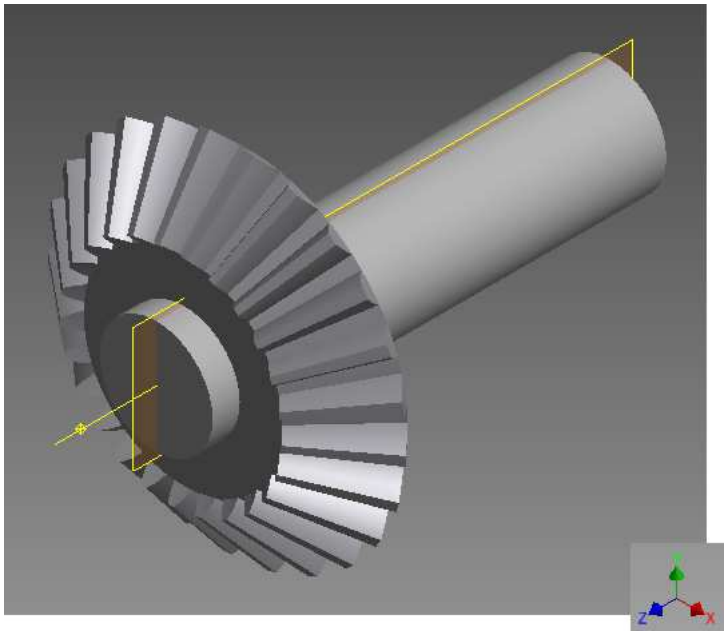
9.2.4.11.1 - Peces sense restringir

Tot seguit es fa coincidir l'eix central de les Z.



9.2.4.11.2 - Restricció de l'eix de les Z

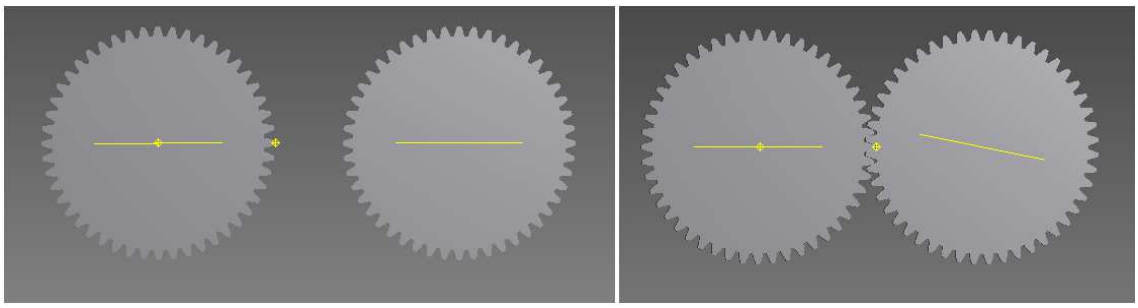
I finalment, es restringeix el pla de les YZ per poder transmetre el moviment de gir i es restringeix a una distància específica, en aquest cas 2 centímetres, la superfície de la base de l'engranatge (la cara més gran) amb la base de l'eix.



9.2.4.11.3 - Resultat final d'uní un engranatge cònic amb un eix

9.2.4.12 - Problemes i solucions d'unió d'elements

Si no es restringís la posició d'un dels elements en el moment d'aplicar les restriccions, ambdues peces es mourien per poder complir les característiques de les restriccions, sense assegurar que la posició final fos l'esperada (per exemple, tinguent l'ordre de les peces girades o els angles invertits).



9.2.4.12.1 - Problema de proximitat amb gir

Les restriccions s'apliquen per proximitat. És a dir, si el centre d'un element s'ha de restringir coincident en un pla, la seva posició un cop restringit serà la posició més pròxima respecte la posició inicial on es trobava el centre de l'element abans de moure. És per això que si no es col·loquen les peces en el quadrant que toca abans d'aplicar la restricció amb el pla auxiliar, la peça es mourà amb el mínim recorregut per complir la restricció, sense assegurar que la posició sigui la desitjada. A més, l'aplicació

d'*Autodesk Inventor* no ens assegura que, al moure un element per restringir-lo, la orientació sigui igual que la original (veure imatge 9.2.4.12.1).

En primeres proves, la unió d'engranatges cilíndrics es feia definint una distància entre els eixos de les Z dels dos elements, en lloc d'aplicar la restricció per contacte actual. El càlcul s'establia agafant la meitat dels dos diàmetres i sumant-los entre si per saber a quina posició s'havia de trobar el segon engranatge. Cal dir que en aquell punt només es treballava en un pla i no hi havien els angles de posició. D'aquesta manera hi havia problemes de graus de llibertat, en què si es connectaven 4 engranatges cilíndrics amb dos eixos creant un circuit tancat (veure prova 2 del tema 10) els elements no rotaven.

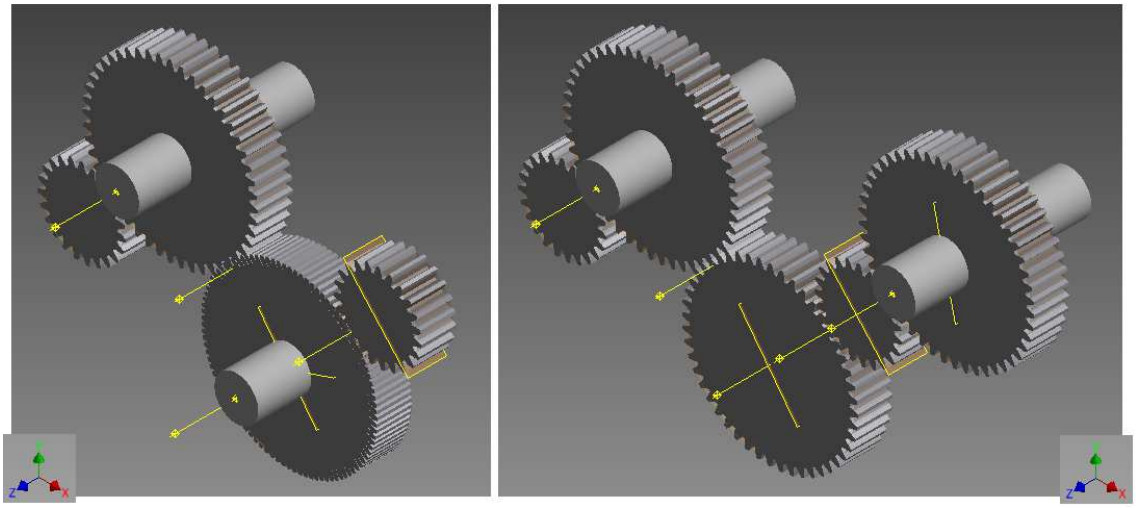
9.2.5 Problemes entre l'aplicació i Autodesk Inventor

S'ha detectat que en alguns casos, sense tenir cap evidència dels motius que ho causen, les operacions de moviment fan que al transportar un element d'una posició a una altra, la posició final no coincideix amb la desitjada, o que els elements ja units en un angle desitjat, saltin i es posicionin en l'angle invers.

Un exemple detectat és, quan es carrega dos cops el fitxer de prova 2, es crea un nou engranatge i s'uneix aquest engranatge amb un engranatge de cada mòdul. Els passos que el fitxer XML segueix són:

- Mòdul 1:
  - Engranatge g1 amb a1 a 5 cm
  - Engranatge g2 amb g1 a 135 graus
- Mòdul 2:
  - Engranatge g1 amb a1 a 5 cm
  - Engranatge g2 amb g1 a 135 graus
- Engranatge as2.g2 amb g1 a 315 graus
- Engranatge g1 amb as1.g1 a 45 graus

El resultat esperat, envers l'obtingut, és:



9.2.5.1 - A l'esquerra, solapament dels elements; a la dreta resultat esperat

Tenim raons per pensar que és un problema propi de l'aplicació d'*Autodesk Inventor*. Concretament pel mètode de translassió utilitzant el *setTranslation,* propi de l'API d'*Autodesk Inventor*, que s'utilitza en el mètode següent:

```
private void makeTranslation(ComponentOccurrence gear1, ComponentOccurrence gear2,
                                                    double posX, double posY) {
     Inventor.Matrix posMatrix = gear1.Transformation;
     double origX = posMatrix.Translation.X;
     double origY = posMatrix.Translation.Y;

     posMatrix.SetTranslation(_invApp.TransientGeometry.CreateVector(origX+posX,
                                          origY+posY,posMatrix.Translation.Z));
     gear2.Transformation = posMatrix;
```

Quan es mou un element i, per exemple, al crear el vector amb els valors (8,8,0), el resultat del SetTranslation no és igual, sinó que queda en diferents valors, per exemple (1.23, 2.47, 0), valor que dista molt de l'objectiu. En canvi, si en el moment d'aplicar aquest mètode, es pausa l'execució i manualment dintre l'aplicació d'*Inventor* es mou l'element a la posició (8,8,0), la posició final serà la desitjada. Llavors la situació és que, en alguns casos, l'aplicació de l'operació *SetTranslation* no aplica correctament la posició final dels objectes, però si es fa manualment dintre l'aplicació sí. Al no aplicar-se correctament la translació, en aquest cas un dels elements salta posicionant-se a l'angle oposat.

En canvi, si mirem la prova 3 del tema 10, en què també es carreguen moduls, un dels quals és del fitxer de prova 2, la col·locació inicial no salta i es manté com es vol.

9.2.6 Tercer botó: *Simulation Drive*

La funcionalitat del tercer botó consisteix en poder simular, de manera visual, el moviment de l'assemblatge generat.

9.2.6.1 - *Simulation Drive:* Botó de simulació

La seqüència que segueix el botó de simulació és la següent:

```
private void buttonExecuteDrive(object sender, EventArgs e) {
     try {
          //Fixar eixos auxiliars
          List<ComponentOccurrence> axisAuxList = parser.getAxisAuxList();
          foreach (ComponentOccurrence occ in axisAuxList) {
               occ.Grounded = true;
          }
          //Eliminar plans auxiliars
          List<ComponentOccurrence> planeAuxList = parser.getPlaneAuxList();
          deleteAuxPlanes(planeAuxList);

          parser.executeDriveConstraint();
     }
     catch (Exception ex) {
          System.Console.WriteLine(ex.Message);
     }
}
```

El procés consisteix en tres punts:

- fixar els eixos auxiliars (per evitar moviments no controlats)
- eliminar els plans auxiliars (per netejar el contingut de l'assemblatge)
- iniciar el moviment del motor (per simular l'assemblatge).

Per fixar els eixos auxiliars s'utilitza la propietat *grounded* del *ComponentOccurrence*. L'altre alternativa és assignar aquesta propietat manualment en el propi *Autodesk Inventor*.

Per eliminar els plans auxiliars s'utilitza la funció *deleteAuxPlanes*, la qual recorre de manera simple la llista d'entrada, eliminant els components de l'assemblatge actual.

Finalment, i més fonamental, es crida la funció *executeDriveConstraint*, pròpia de la classe *ConstraintService*. Per accedir-hi, es crida un mètode de la classe *Parser* amb el mateix nom, que el redirigeix al codi que ve a continuació:

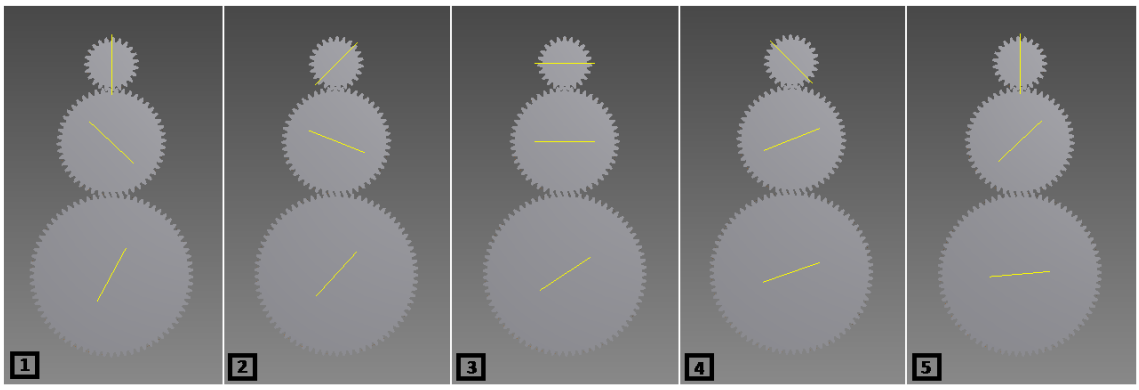
```
public void executeDriveConstraint() {
     try {
          if(motorConstraint != null){
               motorConstraint.DriveSettings.StartValue = "0 deg";
               motorConstraint.DriveSettings.EndValue = "180 deg";
               motorConstraint.DriveSettings.PlayForward();
               motorConstraint.DriveSettings.GoToStart();
          }
          else {
               MessageBox.Show("S'ha d'obrir un assemblatje abans d'executar la
                                                                    simulació.");
          }
     }
     catch (Exception e) {
          System.Console.WriteLine(e.Message);
```

```
        }
}
```

S'utilitza la restricció *motorConstraint*, assignada sobre l'element amb la funcionalitat de motor. Al tractar-se d'un *AngleConstraint*, podem accedir a les propietats del *DriveSettings*, amb les quals podrem simular la funcionalitat de motor.

L'objectiu d'aquestes propietats consisteix en augmentar l'obertura de l'angle de la restricció entre un rang establert. En aquest cas, l'obertura anirà de 0 graus a 180 graus; l'element motor girarà mitja volta.

Podem veure un exemple a la figura 9.2.6.1.1, on es mostra una seqüència de diverses posicions a l'aplicar moviment a l'assemblatge carregat. En aquest cas, s'han creat 3 engranatges cilíndrics, units a 90 graus entre ells, on l'engranatge amb la funcionalitat de motor és el més petit.

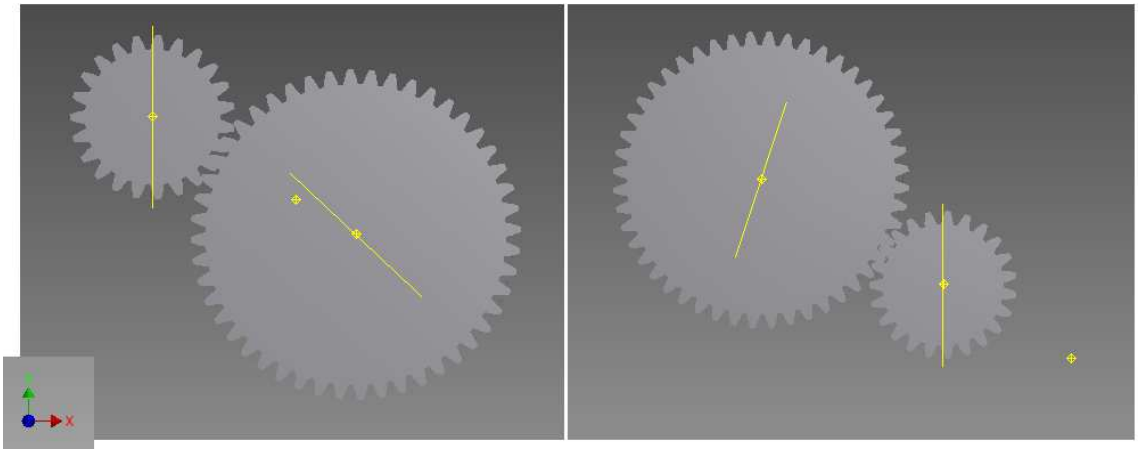


9.2.6.1.1 - Seqüència de simulació de moviment

Tot seguit, amb la propietat *PlayForward* s'inicia el moviment fins finalitzar als 180 graus.

Amb la propietat *GoToStart* es reinicia la posició del motor i, com a conseqüència, la posició de tot l'assemblatge.

El motiu per fixar els eixos auxiliars, i així es justifica la seva utilització, és per assegurar que la posició de tots els elements no varii. En cas contrari, a l'aplicar moviment a l'estructura, es perdria el control de la col·locació relativa de tots els elements, fent que rotessin al voltant de l'element motor o saltessin a la posició inversa respecte els elements restringits.



9.2.6.1.2 - A la dreta, posició correcta; a l'esquerra, l'engranatge gran s'ha invertit

9.2.7 Botó de clausura

Simplement l'acte de clicar el botó per tancar l'aplicació fa enviar un avís a l'*Autodesk Inventor* per tancar-se.

```
if (_started) {
     _invApp.Quit();
}
_invApp = null;
```

10. Resultats

A contunuació hi ha una recopilació de diverses mostres de codis testejats amb l'objectiu de comprovar la funcionalitat de l'aplicació, i els resultats.

En els exemples es seguirà que els noms dels objectes que comencen per "a" són eixos de transmissió, els que comencen per "g" són engranatges cilíndircs i els que comencen per "b" són engranatges cònics.

10.1 Prova 1: Utilització de tots els elements

El codi dissenyat per aquesta primera prova és el següent:

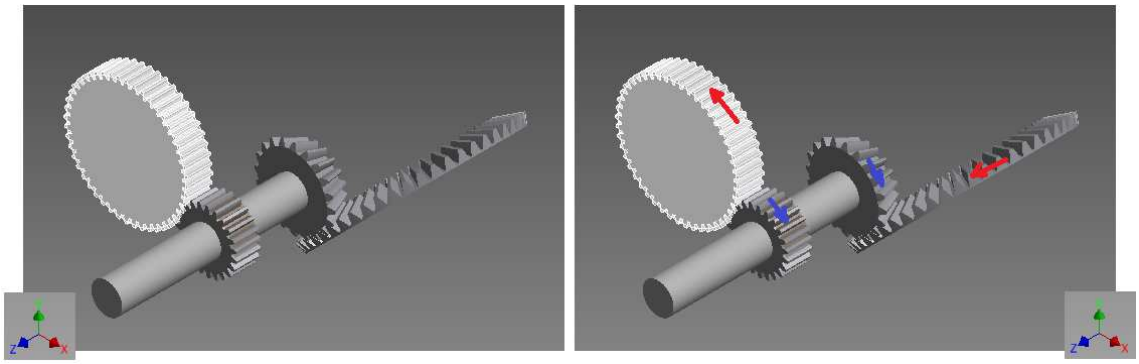
```
<?xml version="1.0" encoding="ISO-8859-1"?>

<assembly xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                        xsi:noNamespaceSchemaLocation="assembly.xsd">
     <creation>
          <name>g1</name>
          <part_info>
               <part>new gear</part>
               <param>10</param>
          </part_info>
     </creation>
     <creation>
          <name>g2</name>
          <part_info>
               <part>new gear</part>
               <param>5</param>
          </part_info>
     </creation>
     <creation>
          <name>b1</name>
          <part_info>
               <part>new bevel</part>
               <param>20</param>
          </part_info>
     </creation>
     <creation>
          <name>b2</name>
          <part_info>
               <part>new bevel</part>
               <param>50</param>
          </part_info>
     </creation>
     <creation>
          <name>a1</name>
          <part_info>
               <part>new axis</part>
               <param>2.5</param>
               <param>15</param>
          </part_info>
     </creation>
     <operation>
          <constraint>join</constraint>
          <param>b1</param>
          <param>b2</param>
          <param>45</param>
     </operation>
```

```
    <operation>
        <constraint>join</constraint>
        <param>g1</param>
        <param>g2</param>
        <param>25</param>
    </operation>
    <operation>
        <constraint>join</constraint>
        <param>g2</param>
        <param>a1</param>
        <param>5</param>
    </operation>
    <operation>
        <constraint>join</constraint>
        <param>a1</param>
        <param>b1</param>
        <param>0</param>
    </operation>
    <motor>
        <motor_name>g1</motor_name>
    </motor>
</assembly>
```

L'objectiu d'aquesta entrada és aconseguir les següents unions:

- Engranatges b2 amb b1 a 45 graus
- Engranatges g2 amb g1 a 45 graus
- Engranatge g2 amb l'eix a1 a 5 cm
- Eix a1 amb l'engranatge b1 a 0 cm
- Motor g1

Aconseguint el següent resultat:



10.1.1 - Resultat prova 1

Amb aquest exemple s'ha posat a prova carregar tots els elements en un únic exemple i moure blocs sencers per poder complir les restriccions. És a dir, quan s'uneix l'eix *a1* amb l'engranatge *b1*, l'eix ja té units a ell els dos engranatges cilíndrics. En una posició de 45 graus, que es manté.

La simulació de moviment també és correcta, expresada amb les fletxes a la imatge 10.1.1 i tinguent com a motor l'element marcat en blanc.

10.2 Prova 2: cicles tancats

Com s'ha comentat amb anterioritat, en primeres proves del projecte s'unien els engranatges establint la distància en què havia d'estar el centre del segon engranatge respecte el primer. Fent que aquest mateix exemple, sense tenir en compte els angles, no simulés correctament el moviment; hi havia un conflicte de graus de llibertat.

Amb els nous canvis s'ha corretgit aquest error.

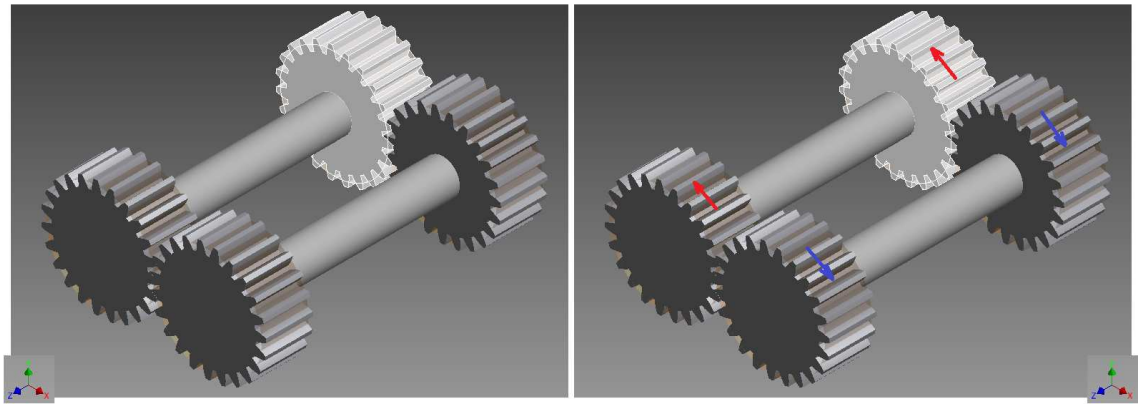
```
<?xml version="1.0" encoding="ISO-8859-1"?>

<assembly xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                        xsi:noNamespaceSchemaLocation="assembly.xsd">
     <creation>
          <name>g1</name>
          <part_info>
               <part>new gear</part>
               <param>5</param>
          </part_info>
     </creation>
     <creation>
          <name>g2</name>
          <part_info>
               <part>new gear</part>
               <param>5</param>
          </part_info>
     </creation>
     <creation>
          <name>g3</name>
          <part_info>
               <part>new gear</part>
               <param>5</param>
          </part_info>
     </creation>
     <creation>
          <name>g4</name>
          <part_info>
               <part>new gear</part>
               <param>5</param>
          </part_info>
     </creation>
     <creation>
          <name>a1</name>
          <part_info>
               <part>new axis</part>
               <param>1.5</param>
               <param>10</param>
          </part_info>
     </creation>
     <creation>
          <name>a2</name>
          <part_info>
               <part>new axis</part>
               <param>1.5</param>
               <param>10</param>
          </part_info>
     </creation>

     <operation>
          <constraint>join</constraint>
          <param>g1</param>
```

```
            <param>a1</param>
            <param>0</param>
      </operation>
      <operation>
            <constraint>join</constraint>
            <param>a1</param>
            <param>g2</param>
            <param>10</param>
      </operation>
      <operation>
            <constraint>join</constraint>
            <param>g2</param>
            <param>g3</param>
            <param>0</param>
      </operation>
      <operation>
            <constraint>join</constraint>
            <param>g3</param>
            <param>a2</param>
            <param>10</param>
      </operation>
      <operation>
            <constraint>join</constraint>
            <param>a2</param>
            <param>g4</param>
            <param>0</param>
      </operation>
      <operation>
            <constraint>join</constraint>
            <param>g4</param>
            <param>g1</param>
            <param>0</param>
      </operation>
      <motor>
            <motor_name>g1</motor_name>
      </motor>
</assembly>
```

L'objectiu d'unió és el següent:

- Engranatge g1 amb l'eix a1 a 0 cm
- Eix a1 amb g2 a 10 cm
- Engranatge g2 amb g3 a 0 graus
- Engranatge g3 amb l'eix a2 a 10 cm
- Eix a2 amb g4 a 0 cm
- Engranatge g4 amb g1 a 0 graus
- Motor g1

I el resultat esperat és el següent:



10.2.1 - Resultat prova 2

El motor està senyalat en blanc i el moviment a la simulació funciona correctament.

10.3 Prova 3: Càrrega de móduls

En aquesta última mostra es carreguen dos mòduls continguts en altres fitxers i s'aplicarà una operació d'unió entre els elements dels mòduls. A més també es crea i s'uneix un element al fitxer d'entrada principal:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<assembly xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                        xsi:noNamespaceSchemaLocation="assembly.xsd">
     <load>
          <name_as>as1</name_as>
          <dir_assembly>
               <dir>prova2-2gears1axis.xml</dir>
               <use_motor>false</use_motor>
          </dir_assembly>
     </load>
     <load>
          <name_as>as2</name_as>
          <dir_assembly>
               <dir>prova1-3Gears.xml</dir>
               <use_motor>false</use_motor>
          </dir_assembly>
     </load>
     <creation>
          <name>b1</name>
          <part_info>
               <part>new bevel</part>
               <param>30</param>
          </part_info>
     </creation>
     <operation>
          <constraint>join</constraint>
          <param>b1</param>
          <param>as1.a1</param>
          <param>0</param>
     </operation>
     <operation>
          <constraint>join</constraint>
          <param>as2.g3</param>
          <param>as1.g2</param>
          <param>45</param>
     </operation>
```

```
     <motor>
          <motor_name>as1.g1</motor_name>
     </motor>
</assembly>
```

Mòdul 1:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<assembly xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                        xsi:noNamespaceSchemaLocation="assembly.xsd">
     <creation>
          <name>g1</name>
          <part_info>
               <part>new gear</part>
               <param>10</param>
          </part_info>
     </creation>
     <creation>
          <name>a1</name>
          <part_info>
               <part>new axis</part>
               <param>2.5</param>
               <param>10</param>
          </part_info>
     </creation>
     <creation>
          <name>g2</name>
          <part_info>
               <part>new gear</part>
               <param>5</param>
          </part_info>
     </creation>
     <operation>
          <constraint>join</constraint>
          <param>g1</param>
          <param>a1</param>
          <param>5</param>
     </operation>
     <operation>
          <constraint>join</constraint>
          <param>g1</param>
          <param>g2</param>
          <param>135</param>
     </operation>
     <motor>
          <motor_name>g2</motor_name>
     </motor>
</assembly>
```

Mòdul 2:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<assembly xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                        xsi:noNamespaceSchemaLocation="assembly.xsd">
     <creation>
          <name>g1</name>
          <part_info>
          <part>new gear</part>
               <param>5</param>
          </part_info>
     </creation>
```

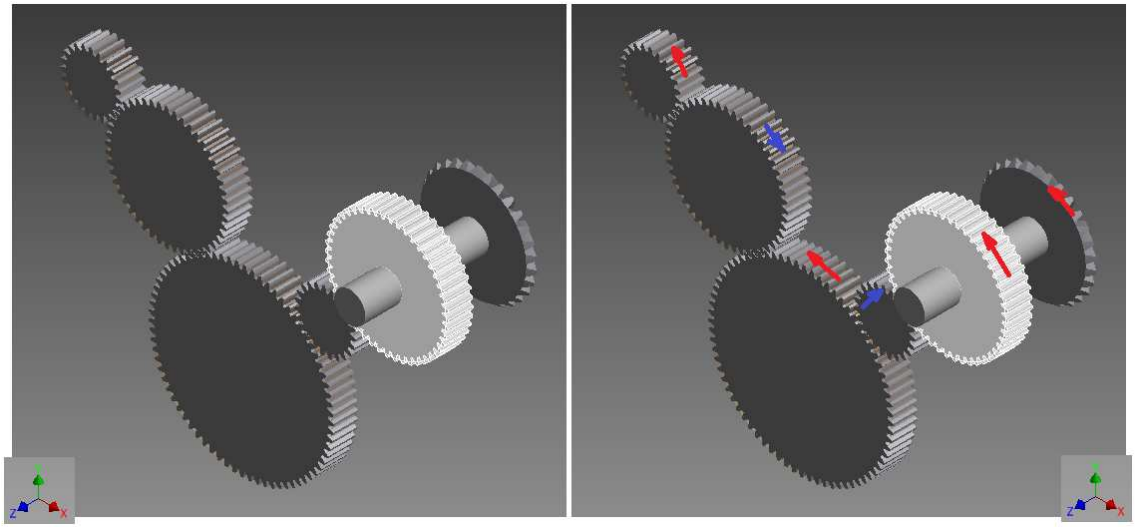
```
    <creation>
        <name>g2</name>
        <part_info>
            <part>new gear</part>
            <param>10</param>
        </part_info>
    </creation>
    <creation>
        <name>g3</name>
        <part_info>
            <part>new gear</part>
            <param>15</param>
        </part_info>
    </creation>
    <operation>
        <constraint>join</constraint>
        <param>g1</param>
        <param>g2</param>
        <param>30</param>
    </operation>
    <operation>
        <constraint>join</constraint>
        <param>g2</param>
        <param>g3</param>
        <param>60</param>
    </operation>
    <motor>
        <motor_name>g1</motor_name>
    </motor>
</assembly>
```

En aquest exemple es posa a prova el fet de carregar assemblatges individuals, moure'ls i treballar amb ells.

Les operacions a realitzar són:

- Càrrega as1
    - Engranatge g1 amb a1 a 5 cm
    - Engranatge g2 amb g1 a 135 graus
    - Motor g2, ignorat
- Càrrega as2
    - Engranatge g2 amb g1 a 30 graus
    - Engranatge g3 amb g2 a 60 graus
    - Motor g1, ignorat
- Engranatge b1 amb as1.a1 a 0 cm
- Engranatge as1.g2 amb as2.g3 a 45 graus
- Motor as1.g1

El resultat obtingut és:



El motor està representat de color blanc. El moviment i càrrega de mòduls funciona correctament.

11. Conclusions

A l'inici del projecte es va fixar l'objectiu de desenvolupar una aplicació que permetés la simulació d'assemblatges compostos per elements mecànics senzills a partir d'un nou llenguatge. Concretament el projecte es va sub-dividir en les següents tasques, on en cada punt s'explicarà la feina assolida:

- Estudi del funcionament d'Autodesk Inventor i la seva API.
    - L'inici del projecte va ser endinsar-se dintre del món d'*Autodesk Inventor*, el glosari amb què treballa, les tecnologies amb què connecta aplicacions externes i com funciona l'API per fer-ho possible. Aquest procés va estar obert i paral·lel amb altres processos a causa de la complexitat i informació que comporta.
- Estudi de la possibilitat d'implementar el projecte amb Python com alternativa al C#. En cas contrari, es continuaria el desenvolupament en C#.
    - A priori, i gràcies al primer punt es tenia constància que *Autodesk Inventor* funcionava correctament amb el llenguatge C#. Tot i així, l'objectiu inicial era poder implementar l'aplicació en el llenguatge Python, cas que finalment no va ser possible. El motiu va ser que no hi havia eines que connectessin aquest llenguatge amb l'*Inventor* específicament. Les eines globals per connectar aplicacions entre si (conegudes com COM) anul·laven operacions vitals per fer funcionar l'aplicació amb l'*Inventor*.
- Adaptació del projecte al nou llenguatge.
    - Aquesta tasca queda anul·lada pel punt anterior.
- Estudi dels graus de llibertat en la transmissió de moviment dels mecanismes.
    - Va ser necessari estudiar com es transmetia el moviment entre elements per poder assegurar la correcta transmissió i simulació dels elements.
- Estudi dels bloquejos de moviment en la simulació automàtica i manual.
    - En primeres versions, en què la unió entre engranatges es feia amb distància, i no per contacte, i en què es treballava en pla, van apareixer problemes perquè els assemblatges quedaven bloquejats o no giraven correctament. Addicionalment es va estudiar fer la simulació amb el *Design Accelerator*, on es simulava el moviment a partir dels graus de llibertat, motiu a més per fer aquest estudi. Gràcies a aquest estudi es varen poder solventar les anteriors problemàtiques i trobar altres alternatives més dinàmiques i àgils
- Desenvolupament del llenguatge per a simulacions automàtiques.

    - S'ha realitzat un nou llenguatge basat en XML per aconseguir la generació d'assemblatges utilitzant els elements descrits.
- Ampliació del llenguatge per a nous components dintre el mecanisme.
    - Inicialment es va desenvolupar pensant en utilitzar engranatges cilíndrics i eixos. Finalment s'han afegit els engranatges cònics, molt més complexos i difícils d'utilitzar.
- Finalització i arrodoniment de la documentació pel treball realitzat.
    - Com a resultat, aquest document.

A més, s'han afegit altres funcionalitats que milloren el resultat final:

- Ampliació del llenguatge per a càrrega d'estructures senceres
    - S'ha afegit la possibilitat de carregar fitxers que contenen, amb estructura en arbre, mòduls d'altres assemblatges a carregar i poder treballar amb els seus elements.
- Posicionament lliure a l'espai
    - Un cop aconseguit un primer prototip on els engranatges i els eixos de transmissió es fixaven en el pla de les XY de l'origen, i veient que aquest mètode comportava errors de graus de llibertat, es va decidir buscar alternatives (eixos i plans auxiliars) que van permetre col·locar els elements lliures a l'espai, sempre tinguent com a referència final l'element motor.
- Simulació de moviment
    - L'objectiu inicial del projecte era generar assemblatges per poder simular manualment el moviment, és a dir, movent amb el ratolí els elements per poder veure la transmissió de moviment entre ells. Finalment s'ha afegit la opció de definir un element motor i aplicar automàticament un gir sobre aquest amb un simple botó. El resultat ha sigut poder simular automàticament el moviment de l'assemblatge dissenyat.

No existeixen eines ni programes coneguts que realitzin les anteriors funcionalitats. Per aquesta raó no hi ha comparació amb altres tecnologies.

12. Treball futur

Tot projecte ha de tenir un temps de maduresa per poder afegir noves funcionalitats i resoldre conflictes que en primeres versions no s'hagin detectat. Aquest projecte no és una excepció.

A dia d'avui s'ha realitzat una bona base per aconseguir treballar amb simulacions d'assemblatges, però hi ha coses a fer, o millorar, per aconseguir millors resultats. Les següents són algunes propostes:

- Generació de fitxers XML
    - La part més costosa d'un usuari a l'utilitzar l'aplicació, sobretot si no és un usuari amb coneixements informàtics, és generar un fitxer XML correcte, amb l'ordre de les estructures i la sintàxi esperada. Es proposa crear un generador XML a partir d'una interfície on l'usuari pugui escullir el contingut del fitxer a guardar/executar.
- Elements de l'assemblatge
    - Es podran augmentar el nombre d'elements que pugui utilitzar el simulador, per exemple, basant-se amb els elements pre-dissenyats de l'aplicació d'*Inventor* que, gràcies a la propietat *Design Accelerator*, s'adaptaran modificant el valor de les variables i fòrmules que continguin.
    - Els *templates,* o objectes per defecte, que utilitza l'aplicació estan basats en figures de la biblioteca interna d'*Autodesk Inventor*. Si volguessim crear nous objectes basats amb tota la biblioteca d'*Inventor,* només obtendríem dues figures més de les que ja utilitzem: un vis sens fi i un engranatge de cremallera. A més, les dimensions dels engranatges d'aquests dos conjunts no són compatibles amb els engranatges que utilitza l'aplicació desenvolupada actualment. Una nova funcionalitat podria ser, modificar els elements anteriors perquè siguin compatibles o generalitzar la càrrega d'elements amb elements propis de l'usuari.
- Simulació dinàmica
    - Es podria afegir dintre el fitxer XML l'opció despecificar les voltes que es vol que fagi el motor i la seva velocitat.

13. Bibliografia

Waguespack, Curtis (2011). *Mastering Autodesk Inventor 2012 and Autodesk Inventor LT 2012* (1ra ed.). Indianapolis: Wiley Publishing, Inc.

Python Software Foundation. (1990-2015). *Python Documentation contents*. Recuperat de https://docs.python.org/2/contents.html

Autodesk Knowledge Network. (2015). *Welcome to Inventor Learning.* Recuperat de http://help.autodesk.com/view/INVNTOR/2014/ENU/

Ekins, B., i Nagy, A. (2015). *Mod the Machine*. Recuperat de http://modthemachine.typepad.com/my_weblog/

Autodesk Inc. (2015). *Mechanism Status and Redundancies*. Recuperat de http://knowledge.autodesk.com/support/inventor-products/learn-explore/caas/CloudHelp/cloudhelp/2014/ENU/Inventor/files/GUID-C7407AC9-CFED-4EBC-9C3E-F6C7DF50AC98-htm.html

Bostongear. *Engineering Information*. Recuperat de http://www.bostongear.com/pdf/gear_theory.pdf

Daycounter, Inc. (2015). *Bevel Gear Calculator*. Recuperat de http://www.daycounter.com/Calculators/Bevel-Gear-Calculator.phtml

14. Annexos

14.1 Requisits per Autodesk Inventor 2015

La informació proporcionada per la següent url conté els requisits mínims per poder instal·lar l'aplicació d'*Autodesk Inventor 2015*:

http://knowledge.autodesk.com/support/inventor-products/troubleshooting/caas/sfdcarticles/sfdcarticles/System-requirements-for-Autodesk-Inventor-2015-products.html

La solució global que proporciona està a la següent taula, extreta directament de l'enllaç:

Requisitos del sistema para Autodesk Inventor 2015 Windows	
Sistema operativo	**Recomendado:** Microsoft ® Windows ® 7 (SP1), Windows 8 o Windows 8.1 de 64 bits [1] [2] **Mínimo:** Microsoft Windows 7 (SP1) de 32 bits [1] [2]
Tipo de CPU	**Recomendado:** Intel ® Xeon ® E3 o Core i7 o equivalente, 3.0 GHz o superior [3] **Mínimo:** Intel ® Pentium ® 4 o AMD Athlon ™ 64, 3 GHz o más veloz o Intel ® o AMD dual core de 2 GHz o más rápido [3]
Memoria	**Recomendado:** [4] de 12 GB de RAM **Mínimo:** 8 GB de RAM para menos de 500 assemblies[4] de pieza
Espacio en disco	**Recomendado:** espacio libre en disco de 250 GB o más **Mínimo:** 100 GB de espacio libre en disco
Gráficos	**Recomendado:** Microsoft ® Direct3D 11 ® o tarjeta gráfica compatible o [5] superior **Mínimo:** tarjeta gráfica compatible con Microsoft ®

	Direct3D 10 ® o [5] superior
Otro	• [6] DE DVD-ROM • resolución de pantalla de 1280 x 1024 o superior • Conexión a Internet para funciones de Autodesk ® 360, descargas de web y acceso a interacción con Subscription • Adobe ® Flash ® Player 10 [7] • Dispositivo señalador compatible con ratón de Microsoft ® • Microsoft ® Internet Explorer ® 8 o superior • Microsoft ® Excel (aplicación de escritorio completamente instalada) 2007, 2010, 2013 para iFeatures, iParts, iAssemblies, personalización de roscas y diseños vinculados a la hoja de cálculo. Office Online no se admite para su uso con Autodesk Inventor. • Microsoft .NET Framework 4.5 SP1 [8]

notas

1. Aplicación de software de Autodesk ® Inventor ® 2015 se proporciona como aplicaciones de 32 bits y de las aplicaciones de 64 bits para instalación y uso en el sistema operativo correspondiente. Autodesk ® Vault Explorer 2015 es una aplicación de 32 bits para instalación y uso en un sistema operativo 32 bits o 64 bits.
2. Autodesk Inventor y Autodesk Vault deben ser el mismo idioma en un equipo determinado. Las versiones en inglés de estas aplicaciones se ejecutarán en los sistemas operativos en idiomas. Otras versiones de idiomas de estas aplicaciones se ejecutarán en sistemas operativos con ese idioma.
3. Autodesk Inventor LT 2015 está pensada para poder aprovechar los conjuntos de instrucciones compatibles con los procesadores Pentium 4, AMD Athlon 64 y AMD Opteron SSE2. Inventor LT 2015 no se puede instalar en equipos que no son compatibles con SSE2. Diversas utilidades están disponibles en Internet que CPUID, incluidos los conjuntos de instrucciones admitidos.
4. Autodesk recomienda una configuración que permita que Microsoft Windows administre la memoria virtual según sea necesario. Siempre debe haber al menos el doble espacio libre en disco que de memoria del sistema (RAM).
5. Consulte la información de tarjeta gráfica publicada por Autodesk.
6. Autodesk Inventor LT 2015 sólo está disponible en DVD (o descarga electrónica en determinados casos).Unidad de DVD-ROM no es necesaria si la instalación se realiza usando métodos de descarga de software.
7. Los componentes de aprendizaje multimedia del sistema de Ayuda de Inventor, como la UI Tour de vídeo, lista de comandos y las animaciones interactivas requieren Adobe Flash Player 10. Si no está ya instalado, puede descargarlo desde el sitio Web de Adobe.
8. Proporcionada por el instalador del producto de Autodesk.

9. Windows XP ® Professional de 32 bits no se recomiendan para Autodesk ® Inventor LT ™ o Autodesk ® Inventor

14.2 API d'Autodesk Inventor

La millor manera per poder navegar per l'API d'*Autodesk Inventor* és a partir de l'ajuda inclosa a la documentació descargada juntament amb l'aplicació. Tot i així, podem veure el contingut de tota l'API al model d'objectes inclós a l'enllaç següent:

- *http://adndevblog.typepad.com/files/inventor2015objectmodel.pdf*

Les dimensions de la imatge són massa grans per poder apressiar en el paper imprés què hi ha escrit als requadres. La millor manera per poder veure el contingut és a partir de visualitzar l'enllaç en una pantalla d'ordinador. Igualment podem veure a la imatge 14.2.1 el model d'objectes per *Autodesk Inventor 2015*.

//Document adjunt, en DIN-A3, amb el nom de "inventor2015objectmodel.pdf".

14.3 Fòrmules dels models utilitzats

En aquesta versió del projecte s'han utilitzat tres tipus d'elements: engranatges cilíndrics, cònics i eixos de transmissió. Els models per defecte inclosos en els fitxers .ipt han estat modificats per poder re-dimensionar els elements, i així obtenir els objectes desitjats, amb les mínimes operacions possibles. Per fer-ho, s'han modificat les fòrmules i valors que contenent aquests elements dintre la biblioteca d'*Autodesk Inventor*.

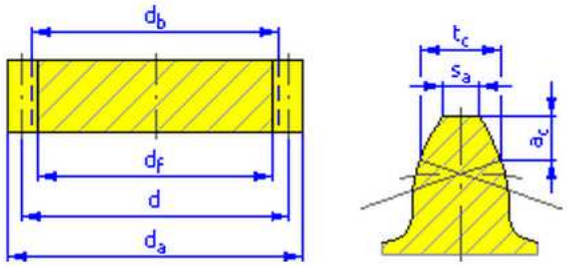
Engranatge cilíndric

Les fòrmules per aquest element s'han modificat per aconseguir re-dimensionar els engranatges cònics modificant només els valors del diàmetre i el nombre de dents desitjat.

Nom de la variable	Equació original	Equació modificada
d0	da_da	
d1	da_b	
d2	0 gr	
d3	da_dw	
d5	0 gr	
d16	da_aw	
d17	0,0 mm	
d34	180 gr / da_z	
d39	0 mm	
da_z2	57,000 su	
da_z	23,000 su	
da_straighttooth	1 su	
da_righttooth	0 su	
da_lefttooth	0 su	
da_dw	46,000 mm	da_d
da_df	41,000 mm	da_da - 9 mm
da_da	50,000 mm	
da_d	46,000 mm	da_da - 4 mm
da_beta	0,000 gr	
da_b	20,000 mm	
da_aw	80,000 mm	
d41	0 mm	
d43	da_aw	
d46	da_aw	
d47	0,0 mm	
d48	0,0 mm	

La variable "da_z" correspon al nombre de dents i la "da_da" correspon al diàmetre extern de l'engranatge. Ambdues variables les assigna l'aplicació.

L'aplicació d'*Inventor* proporciona l'esbós d'algunes cotes que utilitza (veure imatge 14.3.1).



14.3.1 - Cotes engranatge cilíndric

Engranatge cònic

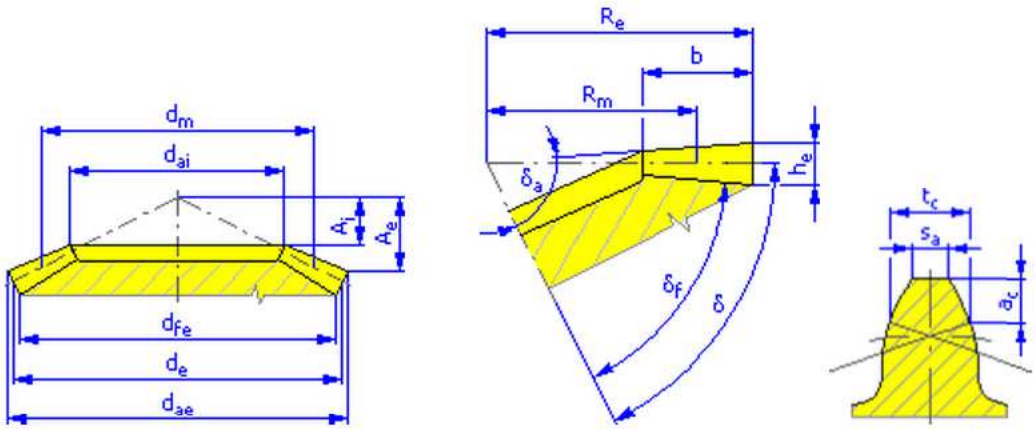
Les fòrmules per aquest element s'han modificat per aconseguir re-dimensionar els engranatges cònics proporcionant els valors del nombre de dents, tan propis com del component amb qui anirà connectat.

Nom de la variable	Equació original	Equació modificada
Paràmetres del model (no llistats)	70 variables sense modificar	
Paràmetres de referència		
d56	25,593 mm	
Paràmetres de l'usuari		
da_b	19,50000000 mm	
da_de	171,00000000 mm	da_z * da_m
da_dae	173,24516421 mm	da_de + 2 mm * 3 su * cos(da_delta)
da_dfe	168,30580295 mm	da_de - 4 mm
da_le	92,19815616 mm	da_de / (2 su * sin(da_delta))
da_omega	90 gr	
da_beta	0,00000000 gr	
da_delta	68,02549201 gr	atan(da_z / da_z2)
da_alpha	20 gr	
da_he	6,60000000 mm	
da_ve	31,71794870 mm	da_le * cos(da_delta) - da_m * sin(da_delta)
da_z	57 su	
da_z2	23,00000000 su	
fd_dv	da_de / cos(da_delta)	
fd_dva	da_dae / cos(da_delta)	
fd_dvb	fd_dv * cos(da_alpha)	
da_zv	152,32738844 su	da_z / cos(da_delta)
fd_loft	da_b + da_he * 0,05 su	
fd_ratio	1,001 su - fd_loft / da_le	
fd_rr	((fd_dd) ^ 2 su - (fd_dvb	

	) ^ 2 su) ^ 0,5 su * 0,5 su	
fd_angle	91 gr / da_zv	
fd_x1	da_dvw * 0,5 su * sin(fd_angle)	
fd_y1	da_dvw * 0,5 su * cos(fd_angle)	
fd_x2	fd_rr * cos(fd_angle2 + fd_angle)	
fd_y2	fd_rr * sin(fd_angle2 + fd_angle)	
fd_x	fd_x1 + fd_x2	
fd_y	fd_y1 - fd_y2	
fd_angle2	acos(fd_dvb / fd_dd) - da_alpha * 0,05 su	
fd_dd	fd_dva - da_he * 0,8 su	
da_dvw	456,98216533 mm	da_de / cos(da_delta)
da_dm	152,91666657 mm	61,70321633 mm
da_x	0,00000000 su	ELIMINAT
da_m	3,0 mm	
da_teethtype	0,00000000 su	

La variable "da_z" correspont al valor del nombre de dents de l'engranatge a modificar i la variable "da_z2" al nombre de dents de l'element a qui anirà connectat.

L'esbós proporcionat per *Autodesk* es pot veure a la figura 14.3.2.



14.3.2 - Cotes per engranatges cònics

15. Manual d'usuari i instal·lació

15.1 Instal·lació

Pel correcte funcionament del projecte necessitem instal·lar les següents aplicacions:

- Descargar i instal·lar Autodesk Inventor. Seguir les instruccions propies.
  - *http://www.autodesk.com/education/free-software/inventor-professional* gratuït per estudiants universitaris.
  - *http://www.autodesk.com/products/inventor/buy*
- Descargar i instal·lar Microsoft Visual Studio 2013
  - *https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx*

15.2 Entorn de desenvolupament

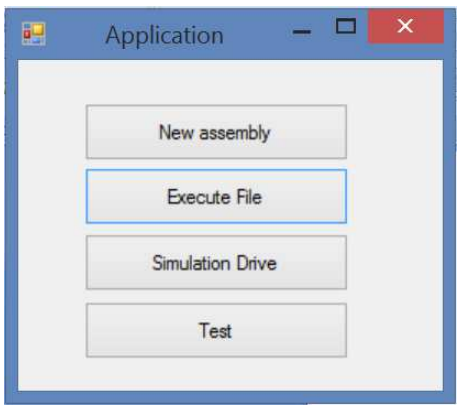
Com que el projecte està pensat per continuar siguent desenvolupat, no hi ha un procés d'instal·lació, sinó d'execució seguint les següents pautes:

1. Copiar la carpeta Assembler, inclós en el disc adjunt, al directori c:\.
2. Obrir el projecte amb un dels mètodes indicats:
   a. Mètode 1: obrir el fitxer Assembler.sln del directori C:\Assembler\Assembler.
   b. Mètode 2: Executar el *Visual Studio* i anar al File->Open->Project/Solution... . Dirigir-se a l'adreça C:\Assembler\Assembler i seleccionar el fitxer Assembler.sln.

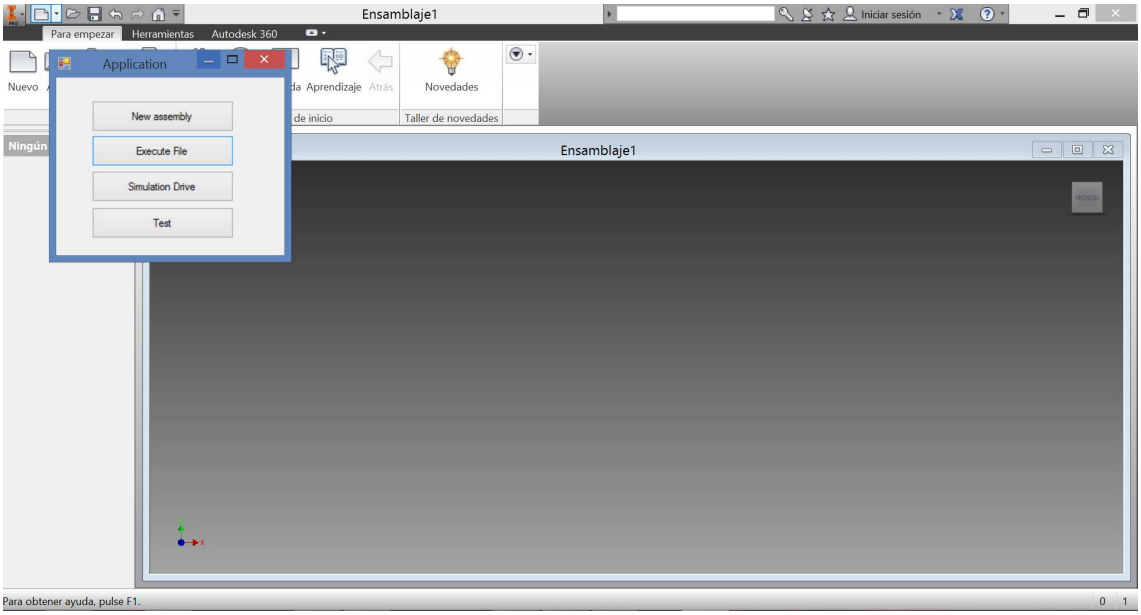
15.3 Execució

Tinguent el projecte obert al Visual Studio, s'inicia el programa amb l'icona 

. Acte seguit, s'obrirà l'aplicació d'*Autodesk Inventor*, en cas que no ho estigués, juntament amb un fitxer d'assemblatge .iam buit, i es mostrarà el menú de l'aplicació.



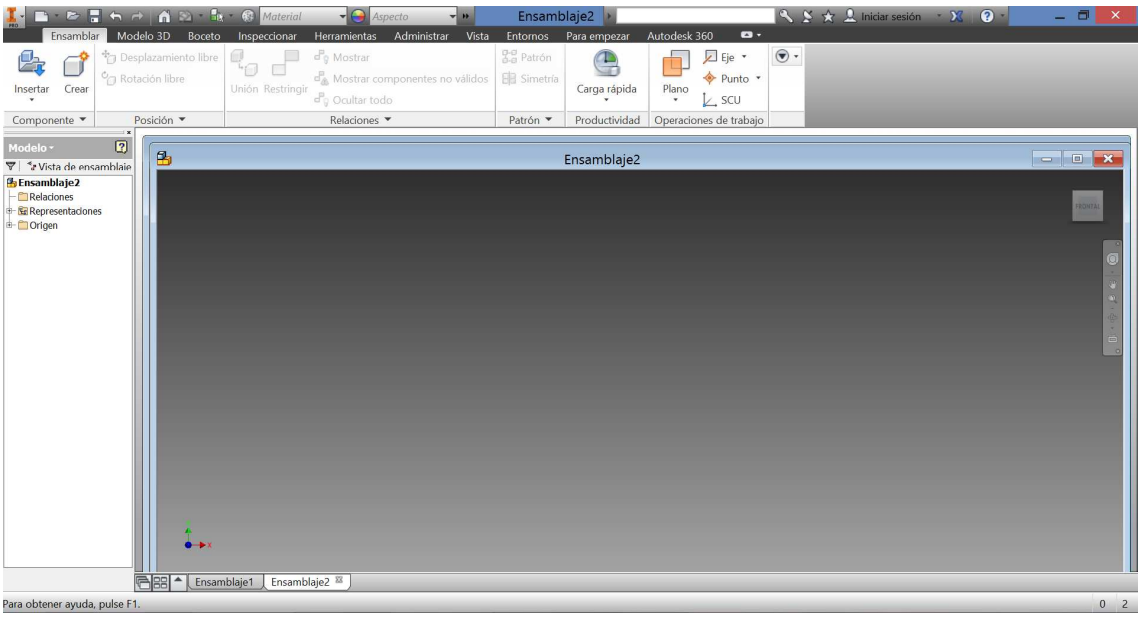
15.3.1 - Menú de l'aplicació



15.3.2 - Inici de les aplicacions

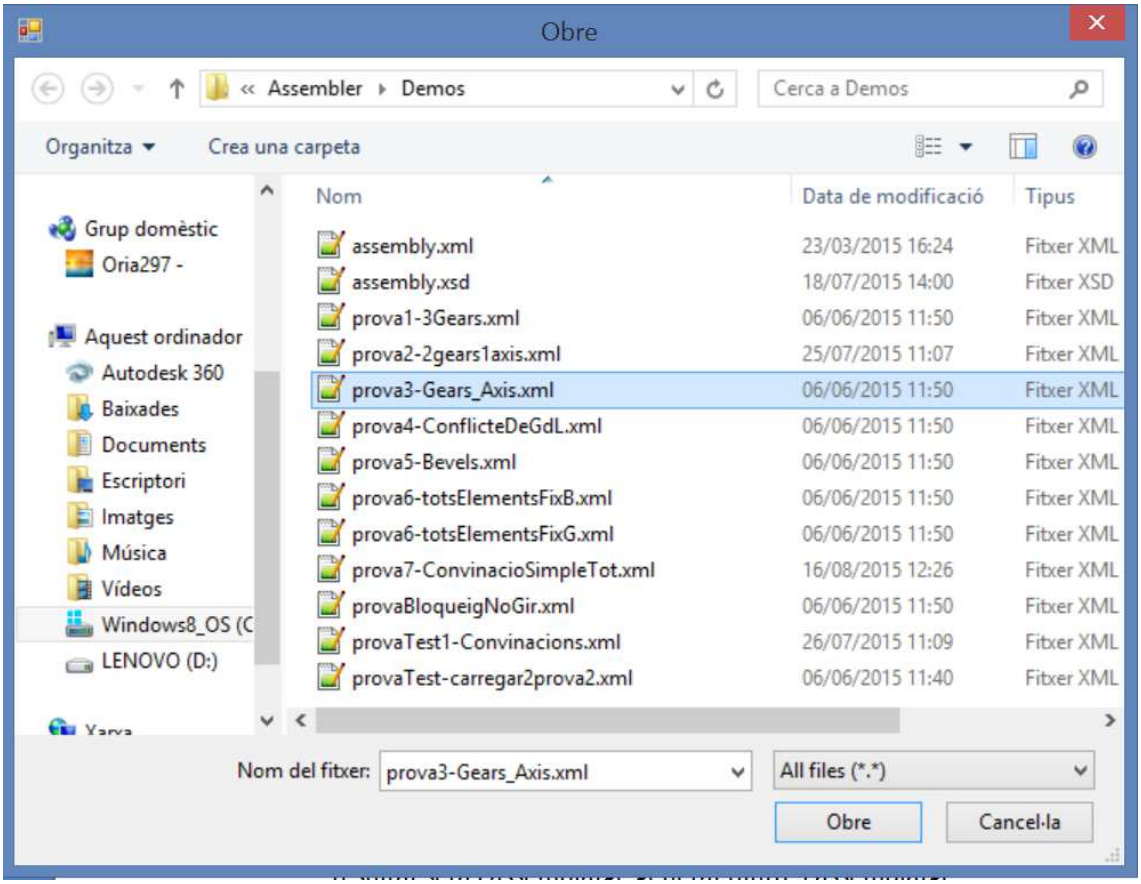
Funcionalitats dels botons:

- Botó "New Assembly": s'obrirà un nou assemblatge buit igual com l'inicial.



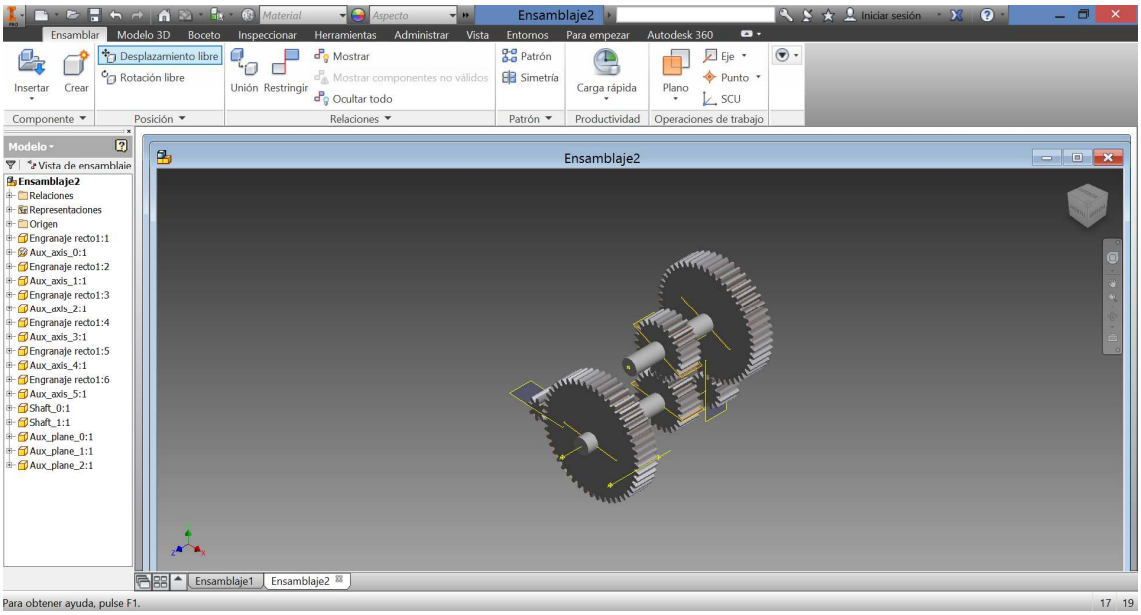
15.3.3 - Nou assemblatge obert

- Botó "Execute File": s'obre el navegador per seleccionar el fitxer d'entrada.



15.3.4 - Carpeta inicial Demo

Un cop seleccionat el fitxer dessitjat, es prem el botó "Obre". El resultat final serà l'assemblatge carregat a l'inventor.



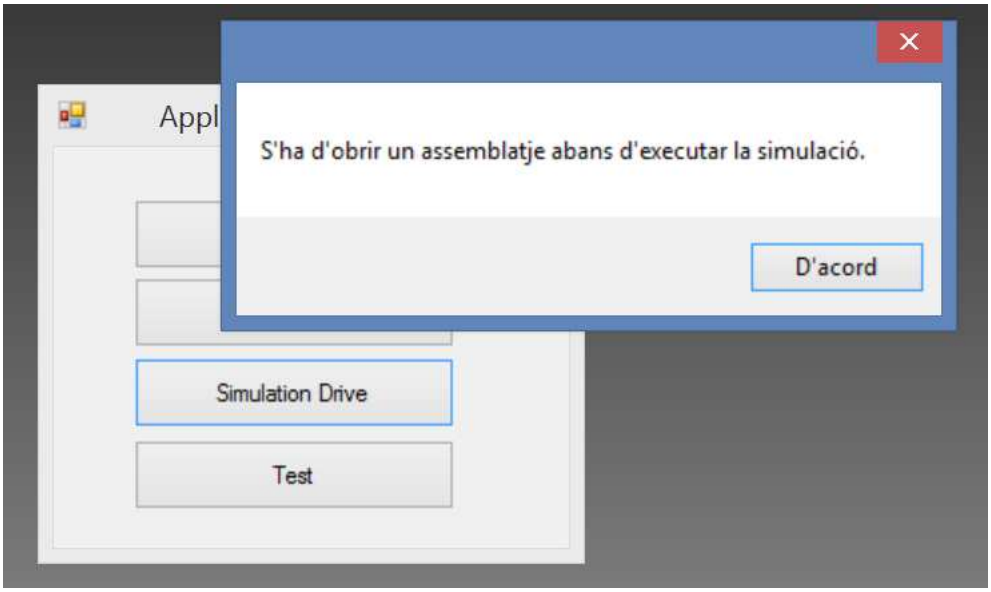
15.3.5 - Assemblatge generat resultant

- Botó "Simulation Drive": la simulació prèviament carregada iniciarà el moviment.
- Botó "Test": sense funcionalitat. No realitza cap acció.

15.4 Alertes habituals

En el procés d'utilització de l'aplicació poden apareixer diverses pantalles d'avís. A continuació es mostren quines alertes són, quan apareixen i la solució proposada.

Falta càrrega de l'assemblatge



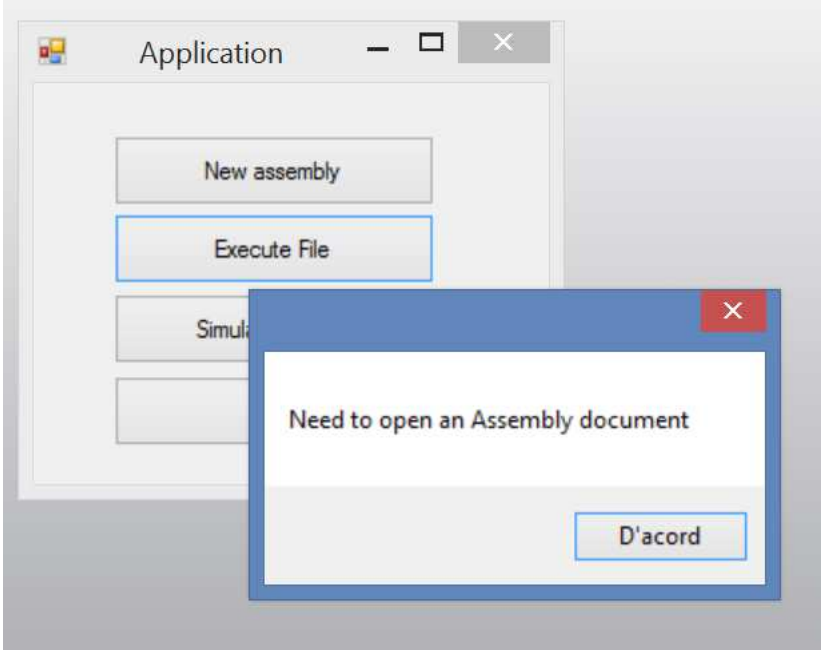
Aquest avís acostuma a apareixer quan s'intenta executar el "Simulation Drive" sense tenir cap assemblatge prèviament carregat o, en cas constrari, l'assemblatge no conté cap element amb la funcionalitat de motor.

Solució proposada: executa un fitxer XML que contingui les etiquetes propies pel motor.

Falta assemblatge buit

L'avís indicant l'absència d'un assemblatge apareix a l'inentar seleccionar un fitxer XML amb l'"Execute File" sense tenir cap fitxer .iam obert a l'*Inventor.*

Solució proposada: executa el botó "New Assembly" abans de seleccionar l'"Execute File".



15.5 Disseny de fitxers XML

Cal saber l'estructura i etiquetes que necessiten els fitxers XML que pot acceptar l'aplicació. Les bases necessàries per dissenyar correctament un fitxer d'entrada XML d'assemblatges es troba al punt 9.1 d'aquesta documentació.