UNIVERSIDAD
NACIONAL
DE LA PLATA

UN ENFOQUE DIRIGIDO POR MODELOS PARA LA CREACIÓN DE SISTEMAS ROBÓTICOS CON MISIÓN PREDETERMINADA

Mattone Nicolas
Montanari Franco

Motivación

La programación de robots presenta dos grandes problemas:

- Necesidad de **personas expertas** en el desarrollo.
  - La arquitectura de un sistema robótico es compleja y variable.
  - El entorno físico es incierto y cambiante.
  - Requieren una mayor atención en comparación a los sistemas de propósito general.

- El proceso de desarrollo no utiliza **ninguna técnica de modelado**.
  - Se pierde la posibilidad de generalizar conceptos.

Objetivo

- Aplicar los conceptos del Desarrollo de Software Dirigido por Modelos para facilitar la programación de sistemas robóticos.

- Desarrollar una herramienta que permita a los usuarios resolver de forma abstracta un problema particular de la robótica.
  - Utiliza un lenguaje gráfico, específico para el problema.
  - Genera el código listo para se ejecutado, a partir del modelo definido por el usuario.

Objetivo

Cuál es el problema?

- Un robot **recorre un espacio conocido** con anterioridad.
- Ese recorrido consiste en llegar a un **conjunto ordenado de ubicaciones** de ese espacio.
- Cuando el robot llega a una ubicación, **puede ejecutar (o no) una serie de acciones**.
- Una vez finalizada la ejecución de la/s acción/es en una ubicación, el robot continúa con el recorrido.

Agenda

1. Conceptos básicos
2. Estado del arte
3. Descripción de tecnologías
4. DSL para robots con misión predeterminada
5. Generación de código
6. Conclusiones

Agenda

1. **Conceptos básicos**
2. Estado del arte
3. Descripción de tecnologías
4. DSL para robots con misión predeterminada
5. Generación de código
6. Conclusiones

Conceptos básicos

*"Los **modelos** son el foco central del **Desarrollo de Software Dirigido por Modelos (MDD)**"*

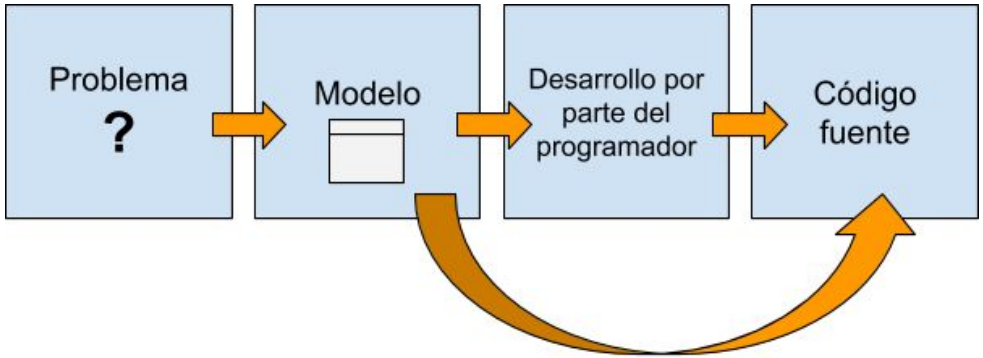
Modelo: representación de un hecho o fenómeno de la realidad.

- Analizar su naturaleza.
  - Elementos que lo forman y sus relaciones.
- Facilitar su comprensión.

Conceptos básicos

MDD es un paradigma de desarrollo software que permite mejorar el proceso de construcción de software, basándose en un **proceso guiado por modelos cuyo motor son las transformaciones entre estos.**

- Mayor nivel de **abstracción**.
- Aumento de confianza en la **automatización** asistida por computadora
- Uso de **estándares industriales**.

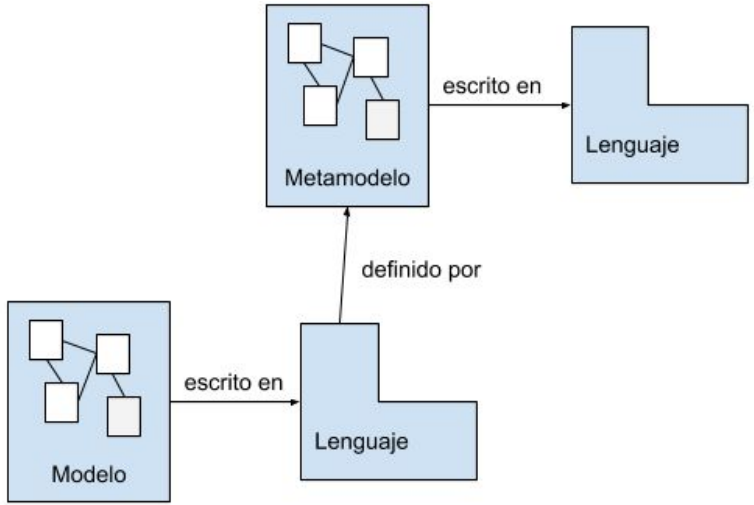


Generación automática del código a través de una herramienta con soporte MDD.

Conceptos básicos

Metamodelado: Permite definir formalmente lenguajes de modelado.

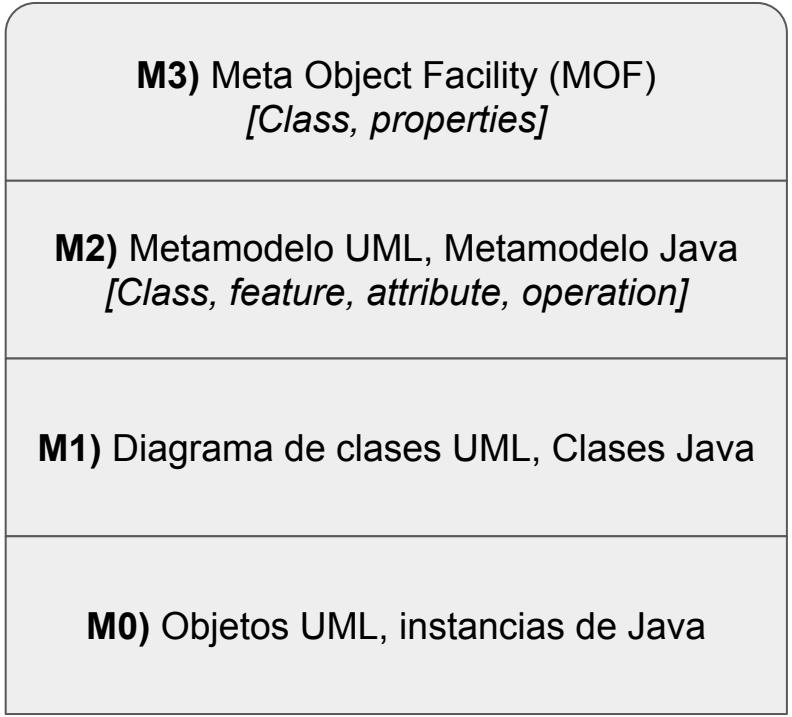
- Metamodelo.
  - Describe qué **elementos** pueden ser empleados para la creación de un **modelo**, y cómo los mismos pueden ser **conectados**.
  - Describe la **sintaxis abstracta**.
- Metalenguaje.
  - Permite describir un lenguaje por medio de un metamodelo.



Conceptos básicos

Arquitectura de cuatro capas de la OMG:

- Nivel **M3**: Meta-metamodelo.
- Nivel **M2**: Metamodelo.
- Nivel **M1**: Modelo.
- Nivel **M0**: Instancias.



Conceptos básicos

Lenguaje específico de dominio (DSL):

- **Propuesta concreta** de MDD.
- Lenguaje focalizado y especializado en un **problema particular**.
- El producto final es **generado automáticamente** a partir de estas especificaciones de alto nivel
- Pueden ser textuales o visuales.

Agenda

1. Conceptos básicos
2. **Estado del arte**
3. Descripción de tecnologías
4. DSL para robots con misión predeterminada
5. Generación de código
6. Conclusiones

Estado del arte

1) ***A Model-Driven Approach To Constructing Robotic Systems.*** *Claudia Pons, Carlos Neil, Roxana Silvia Giandini, Gabriela Pérez, Marcelo De Vincenzi. XVI Workshop de Investigadores en Ciencias de la Computación, 2014.*

- Investiga el uso actual de técnicas modernas de ingeniería de software aplicadas a la robótica.
  - Foco en la generación automática de código.
- Presenta una nueva metodología para programar robots.
  - Definir los comportamientos con un alto nivel de abstracción.
  - Foco en la reutilización y mantenimiento.

Estado del arte

2) ***Component-Based Robotic Engineering - Part I & II.*** *Davide Brugali, Patrizia Scandurra, Azamat Shakhimardanov. Robotics & Automation Magazine, IEEE. 16. 84-96. 10.1109/MRA.2009.934837, 2010 (Part I). Robotics & Automation Magazine, IEEE. 17. 100 - 112. 10.1109/MRA.2010.935798, 2010 (Part II).*

- Aplica la Ingeniería de Software basada en componentes en el desarrollo de sistemas robóticos.
  - Foco en la reutilización y mantenimiento.
- El primer artículo trata sobre los componentes individuales.
- El en el segundo, se discute cómo ensamblar estos componentes.
- Como ejemplo, analiza el problema de *Path Planning*.

Estado del arte

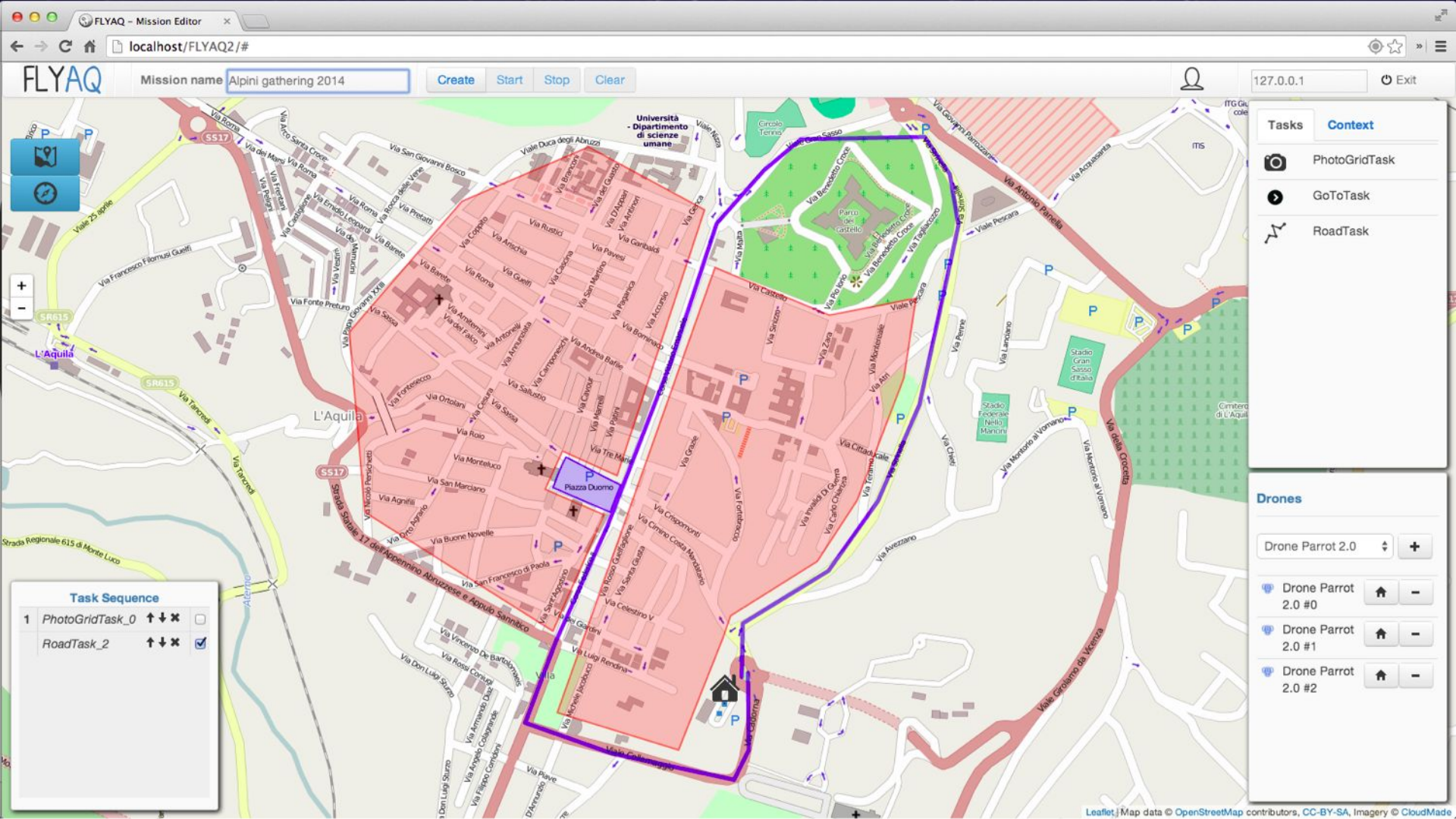
3) ***Towards Easy Robot Programming: Using DSLs, Code Generators and Software Product Lines.*** *Johannes Baumgartl, Thomas Buchmann, Dominik Henrich, Bernhard Westfechtel. 8th International Conference on Software Paradigm Trends (ICSOFT-PT 2013), 2013.*

- Aplica MDD para solucionar el problema de los robots que levantan objetos y los depositan en otro lugar.
  - Mismo objetivo que esta tesina.
- Ejemplo concreto.
  - Panorama de las tecnologías y herramientas.

Estado del arte

4) ***Automatic generation of detailed flight plans from high-level mission descriptions.*** *Davide Di Ruscio, Ivano Malavolta, Patrizio Pelliccione, Massimo Tivoli. Conference: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems. DOI: 10.1145/2976767.2976794, 2010.*

- Plataforma FLYAQ.
  - Permite a los usuarios no expertos definir misiones para drones con un alto nivel de abstracción.
  - Oculta a información relacionada con la dinámica del vuelo de los drones
  - Lenguaje extensible.
- Otro ejemplo concreto.

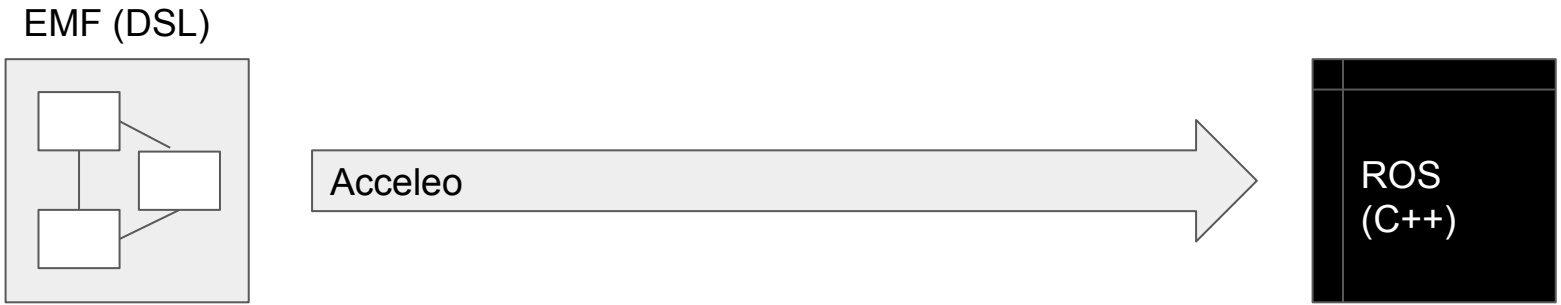

FLYAQ – Mission Editor
localhost/FLYAQ2/#
FLYAQ
Mission name
Alpini gathering 2014
Create
Start
Stop
Clear
127.0.0.1
Exit
Tasks
Context
PhotoGridTask
GoToTask
RoadTask
Drones
Drone Parrot 2.0
Drone Parrot 2.0 #0
Drone Parrot 2.0 #1
Drone Parrot 2.0 #2
Task Sequence
1 PhotoGridTask_0
RoadTask_2
L'Aquila
Piazza Duomo
Leaflet | Map data © OpenStreetMap contributors, CC-BY-SA, Imagery © CloudMade

Agenda

1. Conceptos básicos
2. Estado del arte
3. **Descripción de tecnologías**
4. DSL para robots con misión predeterminada
5. Generación de código
6. Conclusiones

Descripción de tecnologías

1. Eclipse Modeling Framework (EMF).
   a. Definición del DSL.
2. Acceleo.
   a. Generación de código a partir del DSL.
3. Robot Operating System (ROS).
   a. Plataforma específica (código fuente).



Descripción de tecnologías

Eclipse Modeling Framework (EMF):

- Framework para modelado.
- **Meta-metamodelo Ecore (MOF).**
  - **Subconjunto de elementos de UML.**
- Enfoque práctico que unifica tecnologías.
  - Un metamodelo puede ser definido utilizando diagramas UML.
  - Persistencia a través de XMI.
  - Generación automática a clases JAVA.

Descripción de tecnologías

Acceleo:

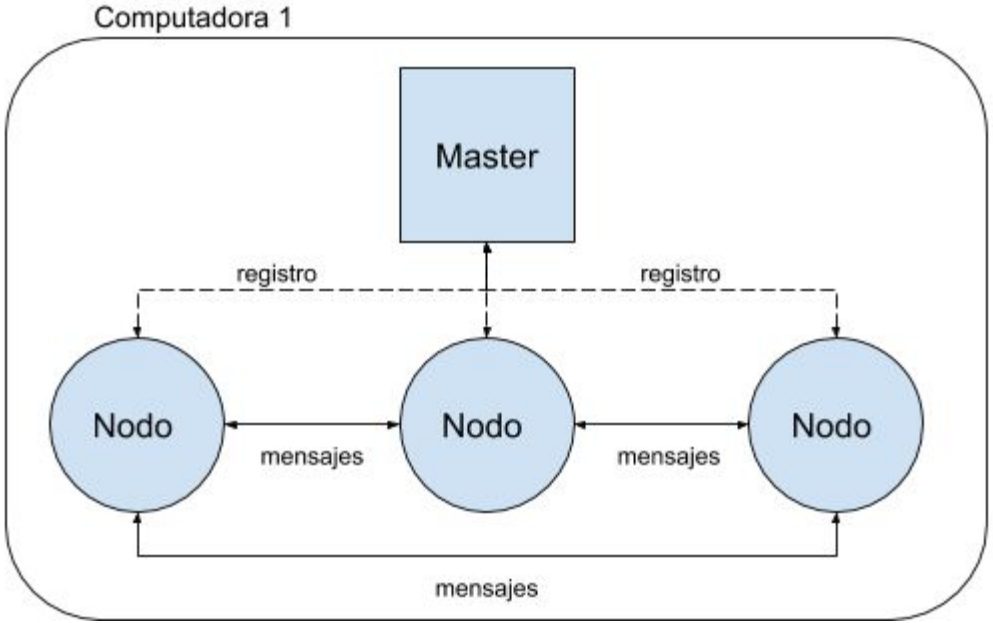
- Generador de código de la plataforma Eclipse.
  - Provee la transformación de modelo a texto.
  - Metamodelo compatible con EMF.
- Basada en un estándar internacional de la OMG.
- Enfoque basado en templates y directivas.
  - Estructura similar a la programación orientada a objetos.
- Sencilla y rápida de utilizar.

```
[comment encoding = UTF-8 /]
 [module generate('http://www.eclipse.org/uml2/3.0.0/UML')/]

 [template public generate(aClass : Class)]
 [file (aClass.name.concat('.java'), false)]
   public class [aClass.name.toUpperFirst()/] {
   [for (p: Property | aClass.attribute) separator('\n')]
     private [p.type.name/] [p.name/];
   [/for]

   [for (p: Property | aClass.attribute) separator('\n')]
     public [p.type.name/] get[p.name.toUpperFirst()/]() {
       return this.[p.name/];
     }
   [/for]

   [for (o: Operation | aClass.ownedOperation) separator('\n')]
     public [o.type.name/] [o.name/]() {
       // TODO should be implemented
     }
   [/for]
   }
 [/file]
 [/template]
```

Descripción de tecnologías

Robot Operating System (ROS):

- **Framework** muy popular para programar robots.
- Desarrollado en el **2007** por el **Laboratorio de Inteligencia Artificial de Stanford (SAIL)**.
  - A partir del 2008, el instituto Willow Garage continúa con el desarrollo.
  - Cuenta con la colaboración de más de 20 instituciones.
- **Middleware**, que provee las funcionalidades básicas de un sistema operativo.
  - Abstracción de hardware.
  - Manejo de sincronización y comunicación.
  - Independencia de lenguaje de programación.
  - Implementación de funcionalidades más utilizadas.


Computadora 1
Master
registro
registro
Nodo
Nodo
Nodo
mensajes
mensajes
mensajes

Descripción de tecnologías

- Un programa de ROS está dividido en **carpetas**.
- Cada una de ellas posee ciertos **archivos** que describen la funcionalidad.
- La unidad básica de la organización del framework se denomina **paquete** o ***package***.
  - **msg/**: *Definición de mensajes (archivos .msg).*
  - **srv/**: Definición de servicios (archivos .srv).
  - **manifest.xml:** *Metadatos del paquete (autor, licencia, dependecias, etc).*
  - **src/**: *Código fuente (C++, Python, Java).*
  - **CMakeLists.txt**: *Directivas de compilación.*

Descripción de tecnologías

```
#include <ros/ros.h>

int main (int argc, char** argv) {
    ros::init(argc, argv, "camara_node");
    ros::NodeHandle nh;

    // Funcionalidad del nodo cámara.

    ros::spin();
    return 0;
}
```

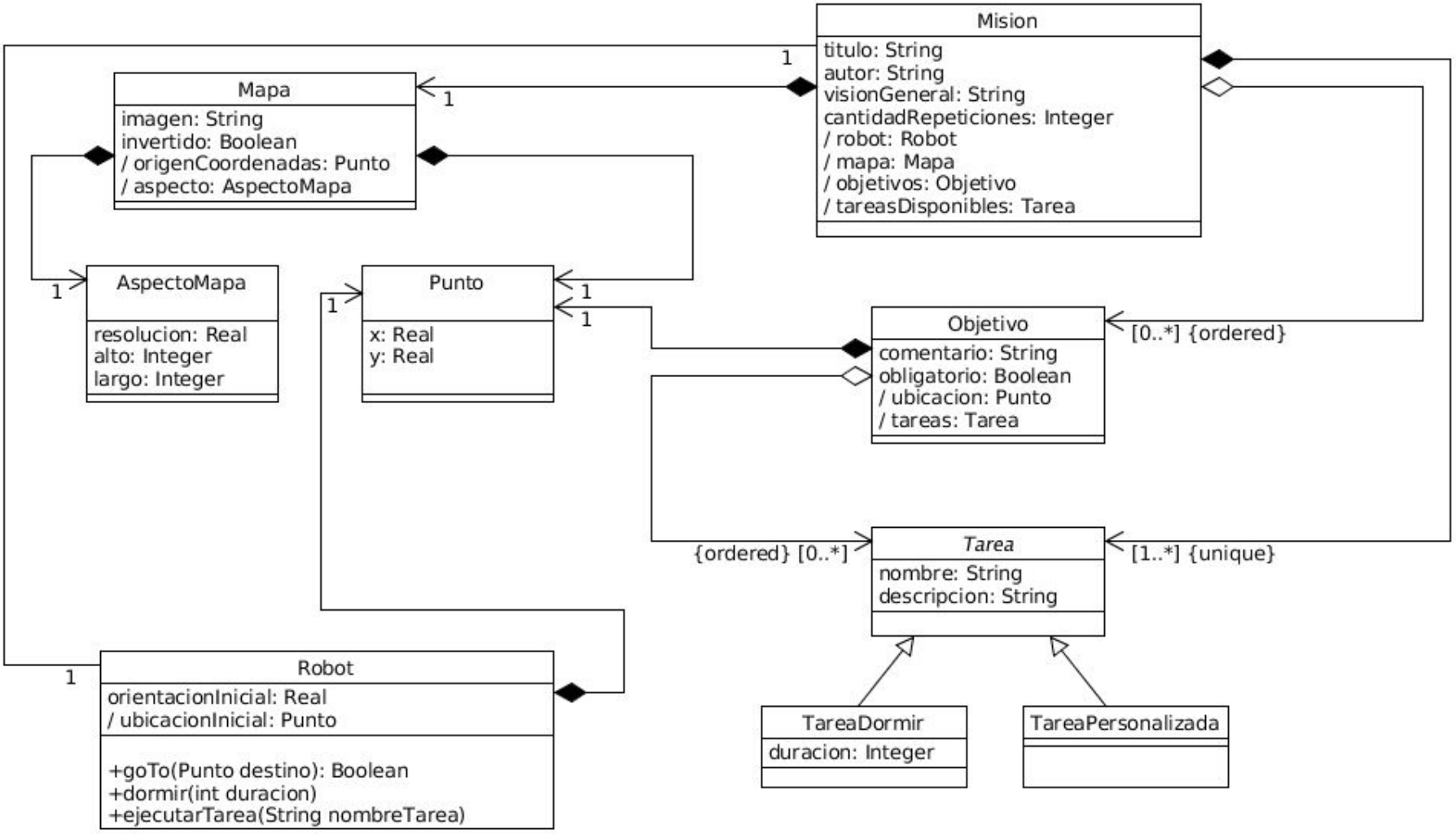
Descripción de tecnologías

¿Por qué utilizar ROS?

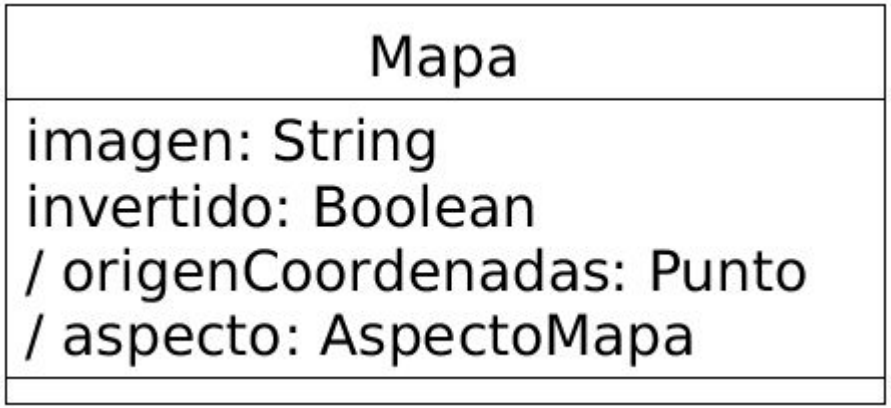
- Gran aceptación por parte de la comunidad.
- Tiene resuelto muchas problemas comunes de la robótica.
  - Sincronización y comunicación.
  - Path Planning (Stack de Navegación).
- Sigue los conceptos de MDD y de los artículos mencionados.

Agenda

1. Conceptos básicos
2. Estado del arte
3. Descripción de tecnologías
4. **DSL para robots con misión predeterminada**
5. Generación de código
6. Conclusiones


Mision
titulo: String
autor: String
visionGeneral: String
cantidadRepeticiones: Integer
/ robot: Robot
/ mapa: Mapa
/ objetivos: Objetivo
/ tareasDisponibles: Tarea
Mapa
imagen: String
invertido: Boolean
/ origenCoordenadas: Punto
/ aspecto: AspectoMapa
AspectoMapa
resolucion: Real
alto: Integer
largo: Integer
Punto
x: Real
y: Real
Objetivo
comentario: String
obligatorio: Boolean
/ ubicacion: Punto
/ tareas: Tarea
Tarea
nombre: String
descripcion: String
Robot
orientacionInicial: Real
/ ubicacionInicial: Punto
+goTo(Punto destino): Boolean
+dormir(int duracion)
+ejecutarTarea(String nombreTarea)
TareaDormir
duracion: Integer
TareaPersonalizada
1
1
1
1
1
1
1
1
[0..*] {ordered}
{ordered} [0..*]
[1..*] {unique}

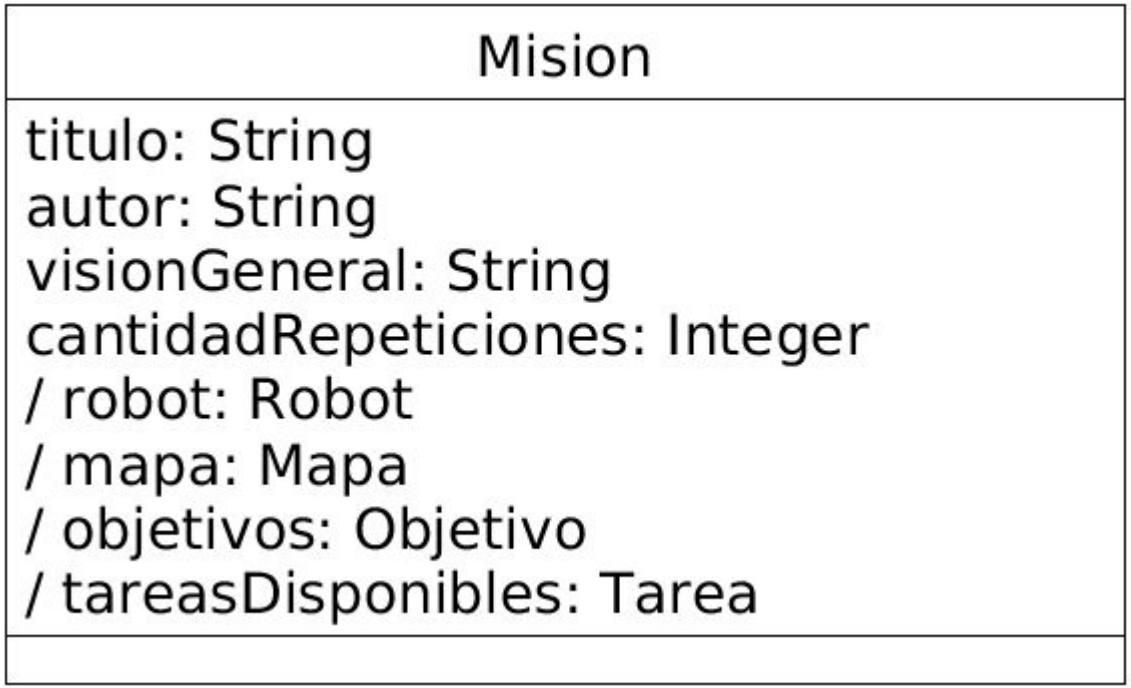
DSL para robots con misión predeterminada



Representación del **mundo real** conocido por el robot, dada por una **imagen**:

- Plano cartesiano.
  - Cada píxel se corresponde con un punto.
- Metro como unidad de medida.
  - Resolución.
- Identificación de obstáculos de acuerdo al color del píxel.

DSL para robots con misión predeterminada



Representa el recorrido predeterminado que debe hacer el robot sobre un entorno conocido.

Contiene toda la información necesaria.

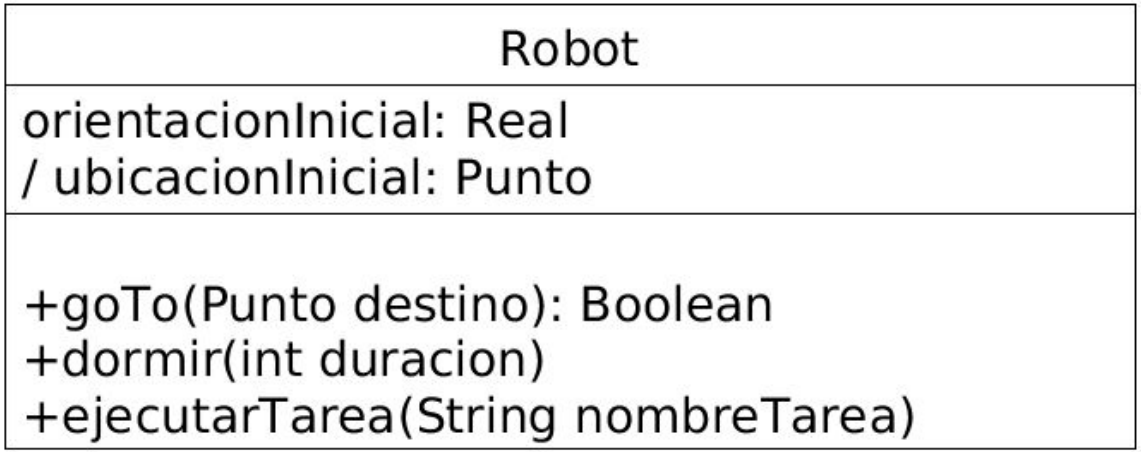
- Mapa.
- Robot.
- Ubicaciones.
- Tareas.

DSL para robots con misión predeterminada

Representa una **posición del mapa** que forma parte del recorrido del robot y la cual **el robot debe alcanzar**.

Una vez que llega a dicha ubicación, **se ejecutan (o no) una serie de acciones**.

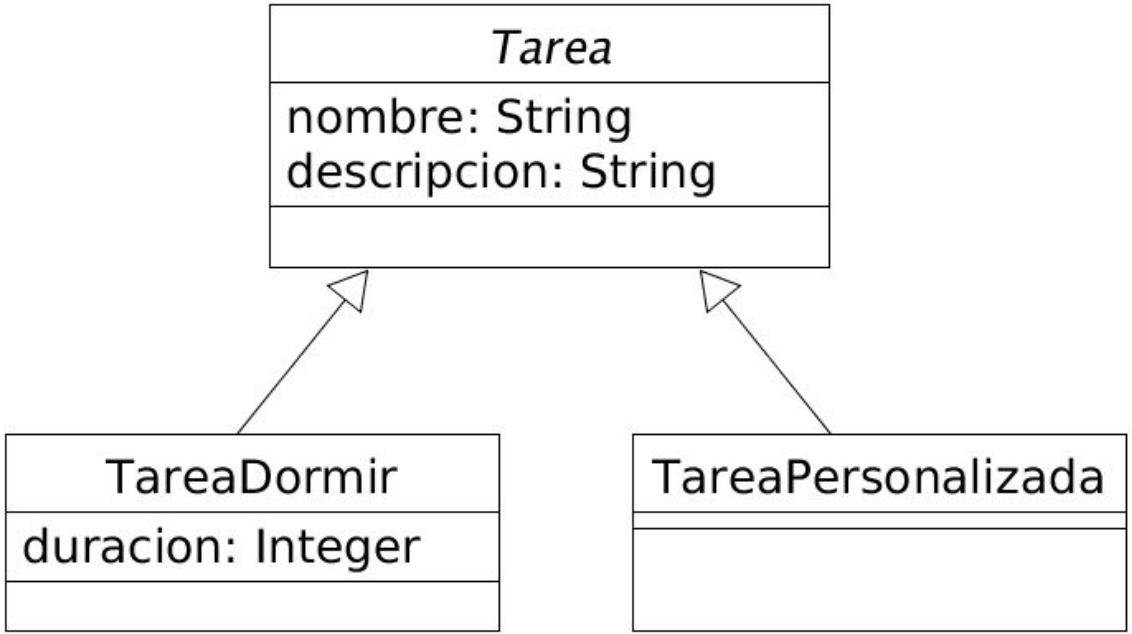
DSL para robots con misión predeterminada



Representa el **robot** que va a ejecutar la misión.

- Sólo contempla las **características comunes** para cualquier robot.
- Deja de lado aspectos específicos.
- **Interfaz necesaria** para llegar a las ubicaciones y ejecutar las acciones.

DSL para robots con misión predeterminada



Representa una **acción a ejecutar** por el robot durante la misión.

- Nombre identificador.
- Tareas comunes a todos los robots.
- Tareas cuya funcionalidad es completada por el usuario (template).

DSL para robots con misión predeterminada

- Sintaxis abstracta.
  - Cómo se define cada elemento del lenguaje.
- Sintaxis concreta.
  - Cómo se representa cada elemento del lenguaje.

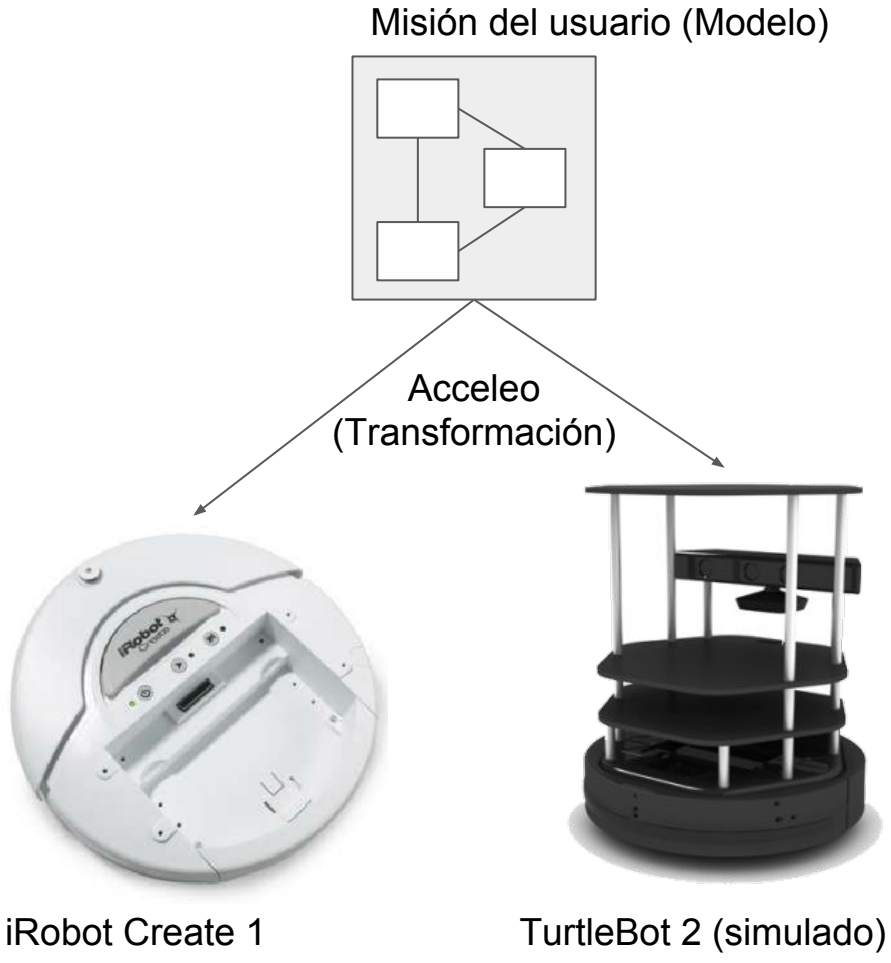
- Herramienta '*Editor de Misiones*'.
  - Permite definir gráficamente misiones.
  - Cada elemento del metamodelo tiene su representación gráfica.
  - Una posibilidad.

Agenda

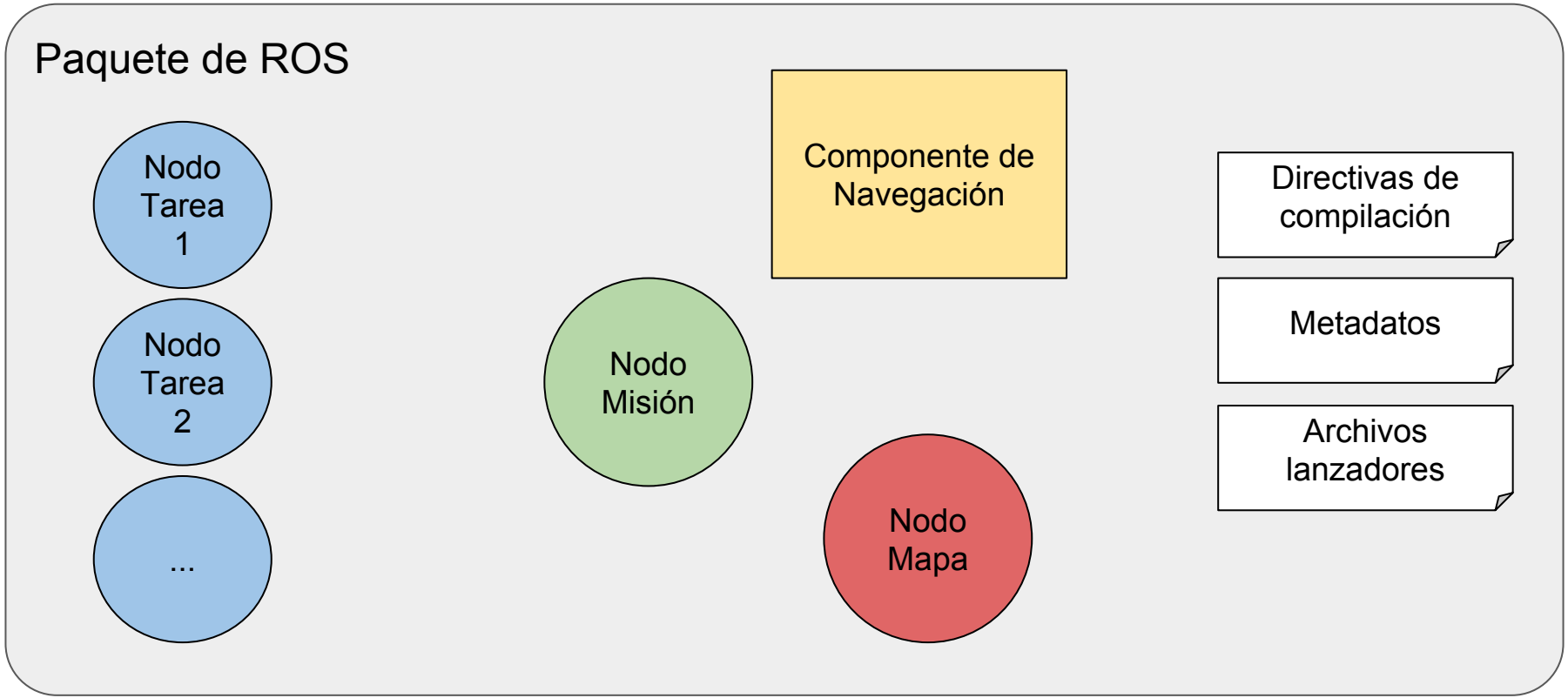
1. Conceptos básicos
2. Estado del arte
3. Descripción de tecnologías
4. DSL para robots con misión predeterminada
5. **Generación de código**
6. Conclusiones

Generación de código

- Módulos de Acceleo.
  - Existe un **módulo principal** (@main), que recibe el modelo y un path a dónde almacenar los archivos generados.
  - Cada módulo **genera un componente de la aplicación** que corre sobre el framework ROS.
  - Código fuente en **C++**.



Generación de código



Generación de código

Componente Mapa:

- Nodo que maneja la estructura de datos que representa el espacio.
- ROS provee un nodo llamado *map_server*.
- La generación consiste en crear el archivo de configuración, que será utilizado para invocar al nodo *map_server*.

```
[template public mapa(aMision : Mision)]
[file (aMision.titulo.concat('/maps/mapa.yaml'), false, 'UTF-8')]
image: [aMision.mapa.imagen /]
resolution: [aMision.mapa.aspecto.resolucion /]
origin: ['['/]-[aMision.mapa.origenCoordenadas.x/], -[aMision.mapa.origenCoordenadas.y/], 0.0[ ']'/]
[if (aMision.mapa.invertido)]
negate: 1
[else]
negate: 0
[/if]
occupied_thresh: 0.9
free_thresh: 0.1
[/file]
[/template]
```

Generación de código

Componentes Tarea:

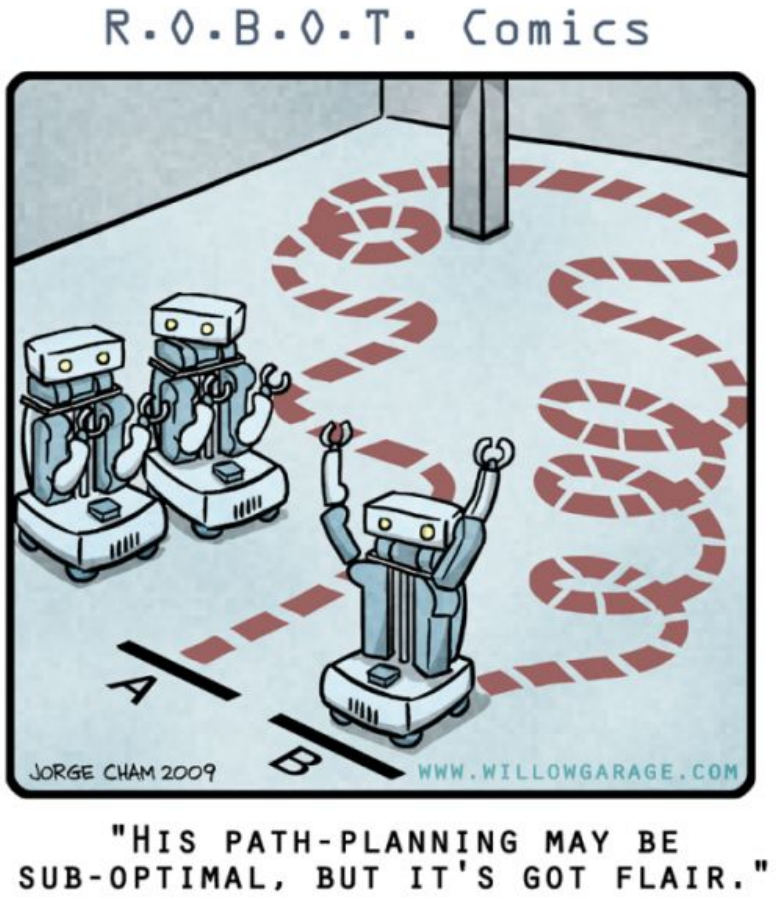
- Por cada TareaPersonalizada definida por el usuario se crea un nodo.
- Cada nodo ofrece un servicio a la topología.
- La identificación del nodo y del servicio viene dada por el atributo *nombre*.
- La TareaDormir es una función nativa del framework (sleep).
- La generación consiste en definir un archivo fuente C++ (cpp).

```
int main(int argc, char **argv) {
    ros::init(argc, argv, "[tarea.nombre/]_srv");
    ros::NodeHandle n;

    ros::ServiceServer service = n.advertiseService(
        "[tarea.nombre/]_srv", service_callback);

    ROS_INFO("La tarea '[tarea.nombre/]' esta disponible.");
    ros::spin();
}

bool service_callback(std_srvs::Trigger::Request &req,
    std_srvs::Trigger::Response &res) {

    //[protected ('for the body of '+ tarea.nombre)]
    // TODO: Completar la funcionalidad de la tarea.
    //[/protected]
    return true;

}
[/file]
```

Generación de código

Componente de navegación:

- Resolver el problema de *Path Planning*.
- Implementa la función *goTo* de la interfaz.
- Precisa de la ubicación y orientación inicial del robot.
- De acuerdo al robot, la generación varía:
  - Stack de Navegación de ROS (requiere configuración).
  - Implementación (archivos .cpp).



Generación de código

Componente Misión:

- Nodo que se encarga de coordinar la ejecución de la misión.
- La generación consiste en un archivo fuente C++ (.cpp).

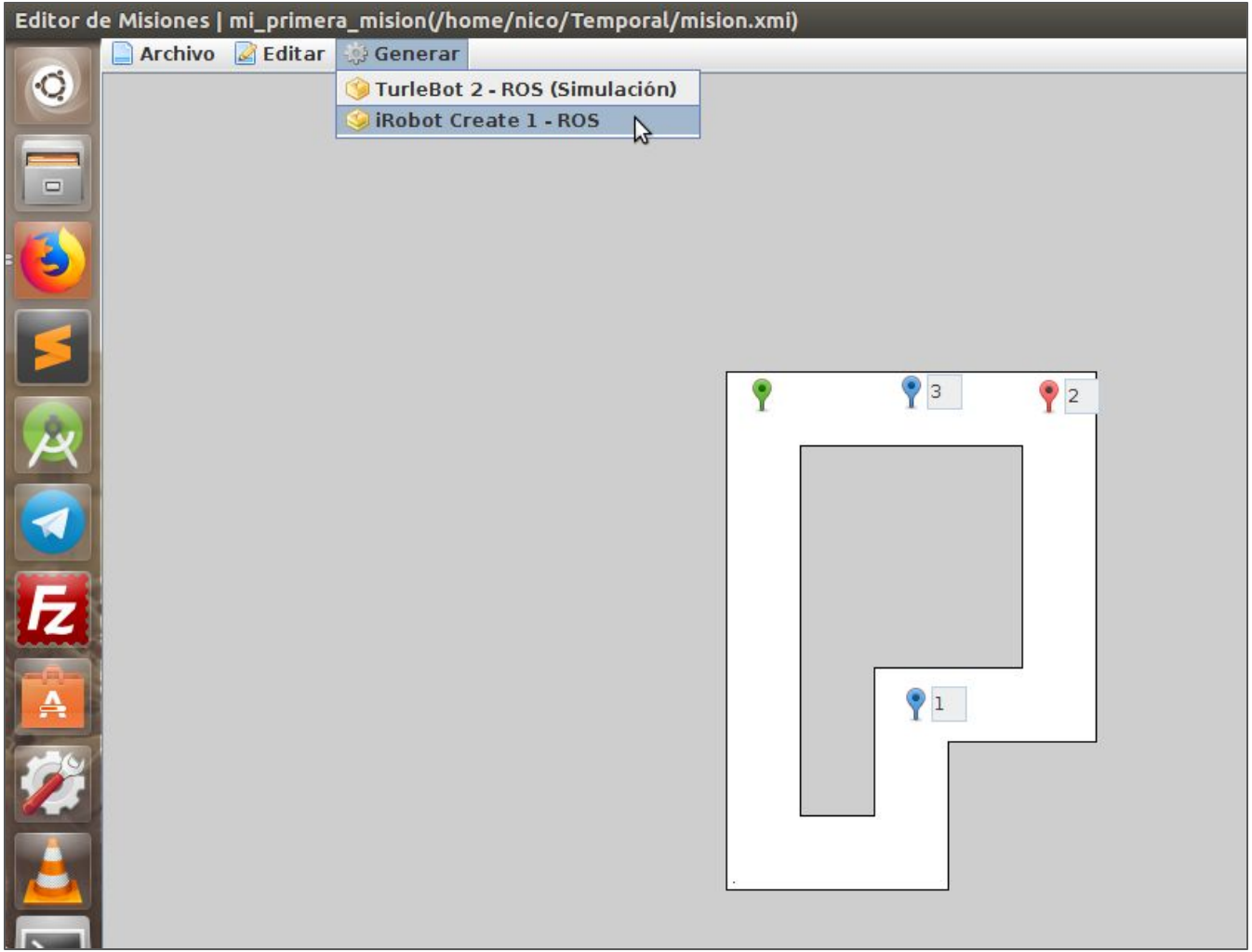
```
for( int x = 0; x < [aMision.cantidadRepeticiones/]; x++ ) {
[for (objetivoActual : Objetivo | aMision.objetivos)]
    [procesarObjetivo(objetivoActual, aMision)/]
[/for]
}
```

```
[template public procesarObjetivo(aObjetivo : Objetivo, aMision : Mision)]
/*
    [aObjetivo.comentario/]
*/

goal_reached = move_to_goal([aObjetivo.ubicacion.x/], [aObjetivo.ubicacion.y/]);
if(goal_reached) {
    ROS_INFO("El robot ha llegado al punto ([aObjetivo.ubicacion.x/], [aObjetivo
    [procesarTareasObjetivo(aObjetivo)/]
```

Generación de código

Otros archivos:

- Manifest.xml
  - Metadatos del paquete, como el autor, la licencia, dependencias, etc.
- Directivas de compilación.
  - Cada nodo es agregado a la lista de ejecutables a compilar.
- Archivo lanzador.
  - Invocado por el usuario para ejecutar todos los mencionados nodos.


Editor de Misiones | mi_primera_mision(/home/nico/Temporal/mision.xml)
Archivo
Editar
Generar
TurleBot 2 - ROS (Simulación)
iRobot Create 1 - ROS
3
2
1

Agenda

1. Conceptos básicos
2. Estado del arte
3. Descripción de tecnologías
4. DSL para robots con misión predeterminada
5. Generación de código
6. **Conclusiones**

Conclusiones

1) Se logró definir **un nuevo lenguaje específico** para resolver un subconjunto de problemas del dominio de la robótica.

a. Independiente de la plataforma de ejecución.
b. Sencillo y amigable.
c. Extensible y reusable.
d. Presenta **limitaciones:**
    i. No abarca todos los casos, sólo cubre misiones sencillas.
    ii. El usuario tiene que completar el código.
e. Primera aproximación al problema.

Conclusiones

2) Se pudo probar el lenguaje con un **robot real**.

3) Se desarrolló una **herramienta gráfica** (*Editor de Misiones*).

a. Permite a los usuarios definir misiones de forma abstracta.
b. Genera el código automáticamente, listo para ser ejecutado.
    i. Limitado a dos modelos de robots.

4) Aporte de un ejemplo concreto de la aplicación de MDD en general y aplicado al dominio de la robótica.

a. Proyecto open source (metamodelo, módulos Acceleo y herramienta gráfica).
b. Sirve como base para trabajos futuros.

Lecciones aprendidas

- MDD nos permitió explorar un paradigma nuevo.
  - Puede ser aplicado y utilizado de forma práctica.
  - Se obtienen resultados concretos.
- Primera experiencia programando robots.
  - Panorama actual.
  - ROS es una excelente opción como plataforma específica.
- La alternativa Eclipse ha resultado efectiva.
  - EMF y Acceleo resultaron ser una combinación práctica.

Trabajos futuros

- **Extender** el DSL.
  - Definir nuevas tareas, comunes a la mayoría de los robots.
    - Similares a la tarea dormir.
  - Permitir la participación de varios robots.
    - Manejo de concurrencia.
- Implementar **generadores de código para otras plataformas específicas** diferentes a ROS.
- Reutilizar el metamodelo del DSL en un **problema más específico**.
  - Restringiendo el uso de tareas personalizadas.
  - El usuario no escribe ni una sola línea de código.

UNIVERSIDAD
NACIONAL
DE LA PLATA

Gracias

Mattone Nicolas
Montanari Franco