



RobMoSys

H2020—ICT—732410

RobMoSys

**COMPOSABLE MODELS AND SOFTWARE
FOR ROBOTICS SYSTEMS**

**DELIVERABLE D7.2
BUSINESS MODELS FOR THE ECOSYSTEM**

Huáscar Espinoza (CEA), Gaël Blondelle, Philippe Krief (EFE), Susanne Bieller (EUR), Anna Principato (TUM)



THIS PROJECT HAS RECEIVED FUNDING FROM THE *EUROPEAN UNION'S HORIZON 2020 RESEARCH AND INNOVATION PROGRAMME* UNDER GRANT AGREEMENT No. 732410

Project acronym: RobMoSys

Project full title: Composable Models and Software for Robotics Systems

Work Package: WP 7 - Exploitation

Document number: D7.2

Document title: Business Models for the Ecosystem

Version: 1.0

Due date: December 31th, 2017

Delivery date: 22.12.2017

Nature: Report (R)

Dissemination level: Public (PU)

Editor: Huáscar Espinoza (CEA)

Author(s): Huáscar Espinoza (CEA), Gaël Blondelle, Philippe Krief (EFE), Susanne Bieller (EUR), Anna Principato (TUM)

Reviewer: Herman Bruyninckx (KUL)

Executive Summary

This document represents the first annual deliverable of Business Models for the RobMoSys Ecosystem (D7.2), as part of work package “Exploitation” (WP7). This deliverable aims at reporting on a market analysis for RobMoSys and defining a set of business models derived from that market study.

The main topics of this document are:

1. Background on business models, open source business models and licenses management.
2. Preliminary market analysis providing a systematic specification of the needs and value chains of the different types of RobMoSys stakeholders, including tool vendors, integrators, system and OEM providers, certification entities, and standardization bodies.
3. Preliminary definition of business models aligned to the market needs and RobMoSys approach. This is organized into community and product business models, as well as in individual business models targeted by the RobMoSys partners.

This first version of the Business Models for the Ecosystem is focused on establishing the basics and on specifying the available knowledge of markets, stakeholder needs and best business approaches for the RobMoSys partners. This scope will be extended in future annual iterations by adding the feedback from Tier-1 experts coming from the business world and taking into account the return of experience of related projects. Workshops with these actors will start at the second year of the project.

Conclusions of this deliverable are:

- The business models for open source assets provide a good basis for RobMoSys partners. In particular, business models associated to the RobMoSys "products" are defined using Business Canvas. For all of these business cases we have demonstrated what the aggregated business process looks like, and how the stakeholders involved benefit from the future RobMoSys ecosystem and community.
- The potential RobMoSys business cases provide enough support to claim that the future RobMoSys ecosystem has, from a business perspective, a clear and feasible goal.
- Not a single business model is able to address all the needs that may appear inside the RobMoSys ecosystem. Partnering is required to cover all the needs, which is a change with respect to the traditional competition model. .

Content

Executive Summary	3
Content	4
1 Introduction	6
1.1 Scope.....	6
1.2 Document Structure.....	7
2 Background.....	8
2.1 Basics of Business Models	8
2.1.1 Business Model Canvas	8
2.1.2 The BOAT Framework.....	10
2.1.3 e ³ Value Models	12
2.2 What is a Robotics Software Platform?	13
2.3 What is Open Source?	13
2.3.1 Elinor Ostrom Principles	13
2.3.2 What is Open Source Software?	14
2.3.3 Why Do we Need Open Source?.....	14
2.4 Licenses Management	15
2.4.1 What is a License?.....	15
2.4.2 Main Open Source licenses.....	16
2.4.3 Compatibilities & Incompatibilities among licenses	20
2.5 Towards Business-Friendly open source ecosystems	22
2.6 Issues of using Open Source Software in the Machinery Industry	22
3 Preliminary Market Analysis	23
3.1 Market Size and Share	23
3.1.1 Market size in robotic-related domains	23
3.1.2 Market size in robotic software and services	24
3.1.3 Market size on software and modelling tools	24
3.1.4 Market size on robotic software platforms.....	25
3.2 Market Trends.....	25
3.2.1 Time to market & competitiveness	25
3.2.2 Reuse and cross-domain harmonization.....	25
3.2.3 Open solutions	26
3.2.4 Increasing service life with shortening product lifecycles	26
3.2.5 Modularity.....	27

3.3	Stakeholders and their Needs	27
3.3.1	Industry	27
3.3.2	Policy Makers and Standardisation Groups.....	29
3.3.3	Scientific and Research Communities	29
3.3.4	Open Source Communities.....	29
3.3.5	Users of Robotics Systems.....	29
3.4	Opportunities and Risks.....	31
3.4.1	Unavailability of success stories.....	31
3.4.2	Inadequacy of use cases and requirements.....	31
3.4.3	Generic or complex results.....	31
3.4.4	Difficult integration	32
3.4.5	Visibility	32
4	Preliminary RobMoSys Business Models.....	33
4.1	Overall Strategy for RobMoSys Business Models.....	33
4.2	Community Business Models	33
4.2.1	Public financing	33
4.2.2	Private non-profit financing.....	34
4.2.3	Financing by whoever needs improvements	34
4.2.4	Financing with related benefits	34
4.2.5	Financing as internal investment	35
4.2.6	Other financing models	35
4.3	Product Business Models.....	36
4.3.1	Pure Services	36
4.3.2	Leveraged Services	39
4.3.3	OSS Co-Ops.....	42
4.3.4	The Proprietary Free Software Product	43
4.3.5	Dual Licensing.....	46
4.3.6	Infrastructure Provider.....	47
4.3.7	Free Software Leveraged Hardware	49
4.3.8	Pure Free Software Product.....	51
4.4	Business Models per Partner	52
4.4.1	Summary of RobMoSys partners view	52
4.4.2	Questionnaire feedback.....	52
5	Conclusions.....	57
6	References	58

1 Introduction

RobMoSys is about managing the interfaces between different roles (robotics expert, domain expert, component supplier, system builder, installation, deployment and operation) and separating concerns in an efficient and systematic way by making the step change to a set of fully model-driven methods and tools for composition-oriented engineering of robotics systems.

RobMoSys's vision is that of an agile, multi-domain, model-driven European robotics software ecosystem. It will consist of specialised set of players with both vertical and horizontal integration levels, providing both widely applicable software products and software-related services. This ecosystem will be able to rapidly address new functions and domains at a fraction of today's development costs.

Popular **digital platforms in robotics** (such as ROS and i-Cub) pride themselves to unite communities of hundreds of stakeholders, which need to be preserved and strengthen. However, these communities are still rather fragmented, by representing specialized customer groups interested by a specific technology or domain. To strengthen the established platforms, to enable interconnections between them, and definitely also between new ones in other domains (not in the least, the application domains of big potential new end-users), RobMoSys envisions an integration approach built on-top-of, or rather “around”, the current code-centric platforms, by means of the systematic development and application of **model-driven methods and tools that explicitly focus on system-of-system integration, at all levels of abstraction and interaction**, hence not just software code.

From a **business perspective**, changes are expected in the evolution of traditionally linear supply chains into complex, dynamic, and connected value webs (co-creation and collaboration). New models of cooperation that depend on seamless integration of diverse partners require new ways of early involvement of customers and business partners into design- and value-adding processes. Digital platforms represent a key enabler to facilitate greater levels of connectivity, collaboration and co-creation with other businesses.

RobMoSys aims at a disruptive change in the approach to robotic software related business by establishing a **common methodology** based on the use of composable software models and by nourishing an **ecosystem** of methodology-based tool chains to support the implementation of the methodology. To cover the largest possible spectrum of stakeholder groups, we envisage to keep a high degree of **openness**, both of tools and also of assets (e.g. models, patterns and libraries).

This document addresses business-related issues of the RobMoSys ecosystem, particularly focused on an open source and open assets strategy. The proposed business models will be incrementally specified and continuously aligned to market needs. This deliverable represents the first annual iteration of **Business Models for the RobMoSys Ecosystem (Deliverable D7.2)**, as part of work package “**Exploitation**” (WP7). We analyse existing and emerging business models mainly based on open source initiatives relevant in the context of robotics systems.

1.1 Scope

This first version of the Business Models for the Ecosystem is focused on **establishing the basics and on specifying the available knowledge of markets, stakeholder needs and best business approaches** for the RobMoSys partners. This scope will be extended in future annual iterations by adding the feedback from Tier-1 experts coming from the business world and taking into account the return of experience from related projects. Workshops with these actors will start at the second year of the project.

1.2 Document Structure

The remaining of this document is organised as follows:

- Section 2 provides a background on business models, open source business models and licenses management.
- Section 3 presents a preliminary market analysis providing a systematic specification of the needs and value chains of the different types of RobMoSys stakeholders, including tool vendors, integrators, system and OEM providers, certification entities, and standardization bodies.
- Section 4 addresses a preliminary definition of business models aligned to the market needs and RobMoSys approach. This is organized into community and product business models, as well as in individual business models targeted by the RobMoSys partners.
- Section 5 summarises next steps and conclusions.

2 Background

2.1 Basics of Business Models

A business model can be described as the system of actions carried out by a corporation or community in the course of its economic activities. It can be decomposed in a number of building blocks. Depending on the various authors, those building blocks are a bit different, but they cover overall the same concepts. We describe three approaches of business model decomposition, from which we will use selected elements for RobMoSys business models.

2.1.1 Business Model Canvas

The business model canvas [Osterwalder 2010] can summarize all the business-related concept descriptions in the right business context. In addition to the BOAT framework, the Canvas includes the value proposition, cost structure, and revenue streams. Together with the BOAT framework and e3 value models, the canvas covers all business aspects and the details to analyse the business case. The Canvas summarizes these results in one overview.

The Business Model Canvas is a strategic management template for developing new or documenting existing business models. It is a visual chart with elements describing a firm's value proposition, infrastructure, customers, and finances. It assists firms in aligning their activities by illustrating potential trade-offs. An overview of all these aspects is depicted in Figure 1.

The building blocks of the Business Model Canvas consist of:

1. **Customer segments.** Customers are the heart of the organisation. This building block defines the different groups of people or organisations that the business wants to reach with the product they offer. It is relevant to define different groups if the offered value needs to be a separate one, either in content, (consumption) channel, relationships, profitability, or different groups are willing to pay for certain aspects of the objects.
2. **Value proposition.** The building block value proposition describes the bundle of products/objects and services that creates value for the customer segments. It is the reason that customers prefer one business over another. The value proposition provides value through various elements such as newness, performance, customization, getting the job done, design, brand/status, price, cost reduction, risk reduction, accessibility, and convenience/usability.
3. **Channels.** The building block 'channels' describes how the business is communicating with its customers. These channels have different functions, like creating awareness about the products/objects offered, determining the value proposition in negotiations, buying products, delivering products, provide value proposition to the customer, as well as customer support.
4. **Customer relationships.** In order to optimize operations and reduce risks of a business model, organizations usually cultivate buyer-supplier relationships so they can focus on their core activity. Complementary business alliances also can be considered through joint ventures, strategic alliances between competitors or non-competitors.

To ensure the survival and success of any business, companies must identify the type of relationship they want to create with their customer segments. Various forms of customer relationships include:

- *Personal Assistance:* Assistance in a form of employee-customer interaction. Such assistance is performed either during sales, after sales, and/or both.
- *Dedicated Personal Assistance:* The most intimate and hands on personal assistance where a sales representative is assigned to handle all the needs and questions of a special set of clients.

- *Self Service*: The type of relationship that translates from the indirect interaction between the company and the clients. Here, an organization provides the tools needed for the customers to serve themselves easily and effectively.
 - *Automated Services*: A system similar to self-service but more personalized as it has the ability to identify individual customers and his/her preferences. An example of this would be Amazon.com making book suggestion based on the characteristics of the previous book purchased.
 - *Communities*: Creating a community allows for a direct interaction among different clients and the company. The community platform produces a scenario where knowledge can be shared and problems are solved between different clients.
 - *Co-creation*: A personal relationship is created through the customer's direct input in the final outcome of the company's products/services.
5. **Revenue streams**. If the customers are the heart of the business, revenue streams are the arteries of the business. An important question for the revenue stream is: what is the willingness-to-pay of each customer segment for the products offered. There are two kinds of revenue streams: transaction based and revolving funds. A finer division is:
 - *Asset Sale* - (the most common type) Selling ownership rights to a physical good. i.e. Wal-Mart
 - *Usage Fee* - Money generated from the use of a particular service i.e. UPS
 - *Subscription Fees* - Revenue generated by selling a continuous service. i.e. Netflix
 - *Lending/Leasing/Renting* - Giving exclusive right to an asset for a particular period of time. i.e. Leasing a Car
 - *Licensing* - Revenue generated from charging for the use of a protected intellectual property.
 - *Brokerage Fees* - Revenue generated from an intermediate service between 2 parties, i.e. broker selling a house for commission
 - *Advertising* - Revenue generated from charging fees for product advertising.
 6. **Key resources**. The building block 'key resources' describes resources that will be required to create value for the customer. They are considered an asset to a company, which are needed in order to sustain and support the business. These resources could be human, financial, physical and intellectual.
 7. **Key activities**. The most important activities in executing a company's value proposition.
 8. **Key partners**. The key partners are the partners that will provide the knowledge, basic functionality, social networks, for the platform to run smoothly. Partnerships can be formed using strategic alliances, co-optation (a co-operation between competitors), joint ventures, and direct buyer-supplier relationships to secure product deliveries.
 9. **Cost structure**. This describes the most important monetary consequences while operating under different business models.

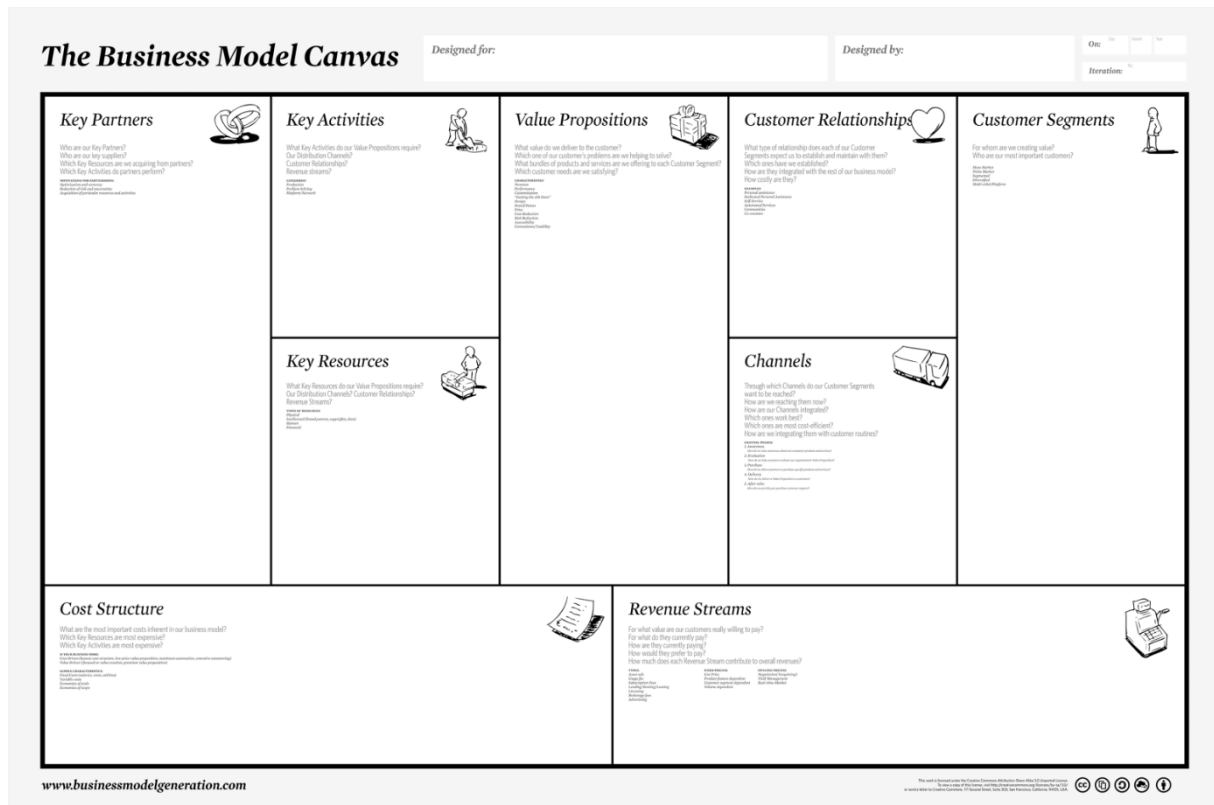


Figure 1: The Business Model Canvas

2.1.2 The BOAT Framework

The BOAT framework and approach [Grefen 2010] assists in developing new e-businesses by integrating several development stages into one spiral model. BOAT is an acronym consisting of the four consecutive development stages or aspects: Business aspect, Organisation aspect, Architecture aspect, and Technology aspect (BOAT). The framework supports working in cycles. After the last step (Technology), a next review round starts in order to demonstrate the impact of the previous choices and to increase the added value of the business proposition. Thus, the approach prescribes a continuous business development. If used rigorously, BOAT can support opening up completely new business opportunities and a big change in existing market value chains, especially in more conventional oriented markets. A more detailed description of BOAT can be found in [Grefen 2010].

In RobMoSys we will include this framework with business, organisation, architecture, and technology aspects in the next iteration of this deliverable.

In the BOAT framework, the business is described in terms of parties, objects and time scope. There are three types of e-business parties: business party (B), consumer party (C) and government party (G). The business objects that can be exchanged between the partners can be categorised as physical goods, digital goods, services, financial goods and hybrid objects. Moreover, in the e-business collaboration scenarios, four types of time scope are represented: static, semi-dynamic, dynamic and ultra-dynamic.

To apply the BOAT framework in RobMoSys, we begin with the business aspect. This describes the business goals of RobMoSys in terms of business drivers, reorganization in business chains, business directions and business structures.

There are a number of concepts in which the business aspect is described:

- Drivers: mainly include reach and richness.
- Chains: mainly include disintermediation and reintermediation.
- Directions: used to conduct business from the viewpoint of one party.
- Structures: used to organize the collaboration of parties in a scenario.
- Model: summarizes the observations made on the different items of the business aspect.

Table 1 lists the concepts for BOAT. Parties included, objects, and time scope are listed in the first column. The second column describes the categories, like B2G – business to government – for the parties' concept. If necessary, a number of additional explanations are added in the last column, called details.

There are three types of e-business parties:

- Business party (B) is a commercial organization of any size and any type, ranging from a multi-national to a one-person company;
- Consumer party (C) is an individual acting as a private person, not on behalf of a business or government;
- Government party (G) is part of a government organization or a related non-commercial organization.

Any initiator-responder combination can be made to characterize the different aspect, from business to business (B2B initiator-responder combination), to government to consumer (G2C).

There are five types of business goods or objects that can be exchanged between the e-business partners:

- Physical goods: tangibles physically exchanged between parties, divided in discrete goods and bulk goods;
- Digital goods: intangibles electronically exchanged between parties, divided into content (copies), information (on demand produced informational data), and software;
- Services: activities that one party specifically performs for another party, either physical or digital;
- Financial goods: sums of money or guarantees to provide money in the future;
- Hybrid objects: a combination of the object classes above.

Four types of time scopes in the e-business collaboration scenarios:

- static: e-business collaboration is long-lasting, or even permanent;
- semi-dynamic: e-business collaborations are changed periodically, but not on the basis of individual orders;
- dynamic: e-business collaborations are determined for each individual e-business order;
- ultra-dynamic: collaborations are changed during the execution of an individual e-business order.

Note that these types can be applied to any of the aspects in the business framework: business, organisation, architecture, and technology.

Table 1. BOAT Business model template

Business model: Template

Concepts	Categories	Details
Parties	-	-
Objects	-	-
Time scope	-	-
Drivers	-	-
Chains	-	-
Directions	-	-

The organization aspect describes how organizations are structured and connected to achieve business goals. It includes business processes and business functions. Since the e³ value models describe the actors and their value exchange, and, with that, the functions and processes involved in the RobMoSys business, this forms an excellent description of the organisational aspect of BOAT. The organisational aspect of RobMoSys is sufficiently covered in the e³ Value Models in Section 2.1.3.

The architecture aspect covers the conceptual structure of automated information systems used to enable organisational structures. It includes information system structures between organizations and within organizations.

The technology aspect describes the technological realization of the systems of with architectural structures. The technology aspect describes software, languages, communication protocols, and hardware.

The business model canvas describes the overall context of the business case in the next section, Section 2.1.1.

2.1.3 e³ Value Models

There is a risk that constructing the business cases might remain as a theoretical exercise if there is no connection to the real added value of the new business. Failing to produce a realistic estimate of the added value, could result in a sudden end of the new venture. For this reason we have included the e³ value model to create a better understanding and estimate of the added value of the RobMoSys ecosystem in the years after the RobMoSys project has ended.

The e³ value methodology helps you to explore your innovative e-business idea - starting from understanding which enterprises and actors are actually involved, to an assessment of profitability for each enterprise. Other models are not as rich or focused on business value as the e³ value model, since they either miss the crucial information required to give a complete picture on a business case level, or they have to include too many details, which makes things more complicated or insecure because of a great number of assumptions need to be made.

As a methodology, e³ value is a graphical approach: you can create your business idea using a number of pre-defined visual elements. Furthermore, e³ value modelling explicitly recognizes that most e-businesses are networks of enterprises. Typically, you develop your e-business idea as an e³ value model in a short timeframe; the methodology has been developed to be tractable and lightweight.

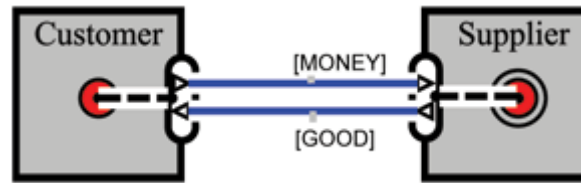


Figure 2: Simple example e3 value model

Figure 2 displays a simple e³ value model showing the value exchange between a customer and a supplier, where the customer pays money in order to get a good. Note that the Customer has the start node (single circle) because he/she takes the initiative in the value exchange. The Supplier is responding to this initiative and because at this level we are not interested in the details of delivering the goods, the activities are represented in an end node (double circle).

2.2 What is a Robotics Software Platform?

By a “robotics software platform” we mean a software package which simplifies programming of several kinds of robotic devices by providing:

- a unified programming environment;
- a unified service execution environment;
- a set of reusable components;
- a debugging/simulation environment;
- a package of “drivers” for most wide-spread robotics hardware
- a package of common facilities such as computer vision, navigation or robotic arm control

The cost of control software accounts for a large share of the overall cost of a typical robotics project. For example, up to 80% of an industrial automation project is spent on system integration which includes software development/customization. So, the main idea behind any robotics software platform is to simplify the job of robotics software engineers – and thus reduce the project cost.

2.3 What is Open Source?

2.3.1 Elinor Ostrom Principles

Elinor Ostrom, Nobel price of Economy in 2009, designed 8 principles for managing stable Common Pool Resource (CPR):

1. **Clearly defined** (clear definition of the contents of the common pool resource and effective exclusion of external un-entitled parties);
2. The appropriation and provision of common resources that are **adapted to local conditions**;
3. **Collective-choice** arrangements that allow most resource appropriators to participate in the decision-making process;
4. **Effective monitoring** by monitors who are part of or accountable to the appropriators;
5. A scale of **graduated sanctions** for resource appropriators who violate community rules;
6. Mechanisms of **conflict resolution** that are cheap and of easy access;
7. **Self-determination** of the community recognized by higher-level authorities; and
8. In the case of larger common-pool resources, organization in the form of **multiple layers of nested enterprises**, with small local CPRs at the base level.

Deciding to contribute to the open source either as a consumer of open source components or producer of open source components or both is a volunteer act to manage **in common** some software code and all the resources attached to this software.

Because the initial investment might feel harder, it is important to feel guided by Elinor Ostrom

principles.

2.3.2 What is Open Source Software?

The Open Source Initiative (<https://opensource.org>) provides a very good definition of Open Source Software (OSS) and defines it in 10 commandments (<https://opensource.org/osd-annotated>):

1. **Free redistribution:** The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.
2. **Include source code:** The program must include source code, and must allow distribution in source code as well as compiled form.
3. **Modifications and derived works:** The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
4. **Integrity of author's source code:** The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.
5. **No discrimination against person and groups:** The license must not discriminate against any person or group of persons.
6. **No discrimination against fields of endeavor:** The license must not restrict anyone from making use of the program in a specific field of endeavor.
7. **Distribution of license:** The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
8. **License not specific to a product:** The rights attached to the program must not depend on the program's being part of a particular software distribution.
9. **License not restricting other software:** The license must not place restrictions on other software that is distributed along with the licensed software.
10. **License technology neutral:** No provision of the license may be predicated on any individual technology or style of interface.

2.3.3 Why Do we Need Open Source?

From Small and Medium-sized Enterprises to large organizations, a lot of companies have adopted and contribute to the one or more open source communities like the Apache Software Foundation, the Eclipse Foundation, and the Linux Foundation. Some of them have been involved for several decades already. This choice has nothing to do with altruism: it is a business strategy.

Actually, organizations like open source software to proprietary software for many reasons, including:

- **Maturity of the model:** There are numerous examples of projects and products based on the OSS which are more reliable and sustainable than other proprietary solutions. Here are a few of them:
 - The Linux operating system, which is now widely adopted by major private or public organizations.
 - Apache HTTP Server, certainly the most used HTTP server. It played a key role in the initial growth of the World Wide Web.
 - The Mozilla web browser called Firefox, which in 2009 was the most popular web

- browser with 32% of the market. In 2016, between 9% and 16% of individuals use Firefox as their desktop browser, making it the second most popular web browser, the first one is Google Chrome.
- In 2001, IBM gave up its Java IDE to the open source community, now known as the Eclipse IDE. After 15 years, this platform supports one of the largest development tool portfolios. Not only Rational, the IBM brand, has built all its desktop workbenches on top of Eclipse, but several SME have based their own development tool on top of the Eclipse Rich Client Platform, the minimal set of Eclipse plug-ins needed to build a rich client application.
 - **Cost of acquisition:** Adopters of OSS obtain a financial gain for each stage of a project. For example:
 - **Free:** it's free to download and use.
 - **Try before buy:** because it is free, companies can try different OSS solutions before making the decision to invest time or resources in a specific one.
 - **Hiring is easier:** because OSS is free, many developers use it and become proficient with the software early on in their career or during their studies. This makes it easier and less expensive to find good developers that have experience with the open source technologies they have adopted for their project.
 - **Training:** it's easier to train a team with the assets produced by OSS community of developers.
 - **Customizability:** open source software can be tweaked to suit various needs. Since the code is open, it's simply a matter of modifying it to add the functionality needed by the project.
 - **Time to Market is shorter:** products don't have to be built from scratch. Companies can rely on sustainable OSS and build their solution on top of it.
 - **Lower total cost of ownership:** companies can rely on OSS community for maintenance and, by joining the community, they mutualized maintenance costs.
 - **Dependence:** Organizations don't depend on the status of the subcontractor who originally built the software. In open source software, if a contributor ceases working on a project for any particular reason, the source code stays accessible and someone else can take over the work.
 - **Quality of the code:** OSS gets closer to what users want because those users can have a hand in improving it.
 - **Security:** Some users consider OSS more secure and stable than proprietary software, mainly because they can control the source code and they can identify and fix errors or omissions. The efforts are mutualized with the other community members, which results in secure and stable source code.

2.4 Licenses Management

2.4.1 What is a License?

A software licence is a legal instrument that specifies the usage or distribution that someone can do from it. Software code is protected in all countries by the authors rights (also known as copyright in the common law environment), exactly the same as another intellectual creation. The software license, even free licenses, are based in authors' rights legislation.

Moreover, in some legislations (USA and Japan), some computer methods could be patented as if they were inventions of industrial use. Software patents are cover by industrial property but in the European Union have no effect. It is important to distinguish the differences between intellectual property and industrial property, because they work in a different way, according to different laws and also have different logic.

The registered names and marks are also regulated by industrial property rights.

In general, WIPO (World Intellectual Property Organization) defines the relation between intellectual creations (inventions, literary and artistic works, the symbols, the names, the images and drawings and the models used in the commerce) and classifies in two categories: industrial property (inventions, patents, marks) and author rights (copyright) [Wipo Web]

To have a better understanding we will present the differences between these three concepts that could be included in the intellectual property denomination:

- **Copyright:** they protect literary, artistic or scientific works of non-authorized uses (copy, modification, distribution, etc.). It includes the computer programs.
- **Trademarks:** It protects the distinguishing signs (logo, graphics, etc.) and the names (including the domain names) of an organization. They try to avoid confusions in commercial names or that two products were called equal. It exists a communitarian (EU) and international system of marks, with laws and courts which solve conflicts, renovation, cession, lapsing, etc.
- **Patents:** they grant exclusive rights for the exploitation of an invention susceptible to be commercialized when the inventor discloses the method or procedure (that is to say, he registers the patent). In Japan and the United States programming methods can also be patented, like for example algorithms. Not thus in Europe, where these patents are not operative because ideas are considered and, until now, the computer programs protect exclusively through the laws of author's rights or copyright.

There are some exceptions to the author's rights (known as fair use in common law) for example, the right for quoting and education, the right for a private copy, the right to information on present subjects, rights of loan for organizations like museums, libraries, etc. These exceptions (or fair uses) vary between one legislation and another one, and in the same they can vary or even be interpreted in different way according to the judge.

The author is considered the person (or the corporate body, in the cases that the law says so) that creates any literary, artistic or scientific work.

The exploitation rights of a work produced by an employee belong to the company. However, the moral rights belongs to the original author (or to his heir)

The free licenses behave to all the effects like any other software license, unless it grants certain liberties to their receiver, instead of to restrict them. Its use is simple, is enough with accepting the terms of the license, it is not even necessary to register the work. The open source and free public licenses (like the ones recognized by OSI [OpenSource Web] or FSF [FSF Web]) are internationally accepted and they do not require any local adaptation.

The free licenses are therefore the legal device to legally implement the four liberties of free software (use, copy, modification and redistribution). The free public licenses for general purposes (like GPL, BSD, GFDL, etc) have the benefits that saves time to the programmer who will not have to spent time writing them and the guaranty the software compatibility under the same license and offer security to programmers and users about their legal consistency.

2.4.2 Main Open Source licenses

All the free licenses grant to all the users the well-known four liberties of free software - use, copy, modification and redistribution. However, they can be classified in two main categories: robust licenses and permissive licenses, based on if they force or not to that the derived works are distributed with the same license (or an equivalent)

Robust licenses (or copyleft): the author allows the modification whenever the new derived work

conserves the same liberties that the original one. They are also one knows as reciprocal licenses, share-alike or share-equal (GPL, GFDL, cc-by-sa).

Permissive licenses: the author solicits to maintain his attribution as it is but he allows any modification without conditions, including that the derived work could be released as privative (BSD, Apache).

GPL

General the Public License, popularly known by its abbreviations GPL, is the legal shape of the concept of copyleft in the computer programs. Written originally by Richard Stallman in 1989 like part of Project GNU, it is the software free license par excellence, the oldest and the most used.

The General the Public License (GPL) of GNU is the most popular and extended free license: between 50% and 70% of the free software are released under the GPL, including the Linux kernel. It was also the first of all of them: it served as model to the ones that came later.

The Free Software Foundation, creator of this license, considered that it had to make sure that all users who obtained a copy of the program received always the same liberties. Therefore, besides the four liberties common to all the free licenses, the GPL incorporates from its first version a clause known as copyleft.

Copyleft requires that the distribution of a modified version conserves identical conditions than the original work. That way, all work derived from software protected by the GPL is guaranteed that it will always conserve the same liberties. Copyleft does not want to restrict rights to the user, but to guarantee that free software under GPL is still free. For that reason this clause is also known as reciprocal license, share-equal or share-alike). Copyleft is then, in practical terms, a form to actively defend the liberties of the users.

GPLv3

From 1991, date in which version 2 of the GPL was published; the technological and social context in relation to computer science has changed enormously. From being something marginal and only known in a few American universities, free software has become a real alternative the industry of traditional software. In addition, the change in the technological context has forced changes in the legislations and new legal conflicts, for example in relation to the programming patents, to DRM, Digitalis Rights Management, also known as Digitalis Restrictions Management)and, generally, to the users rights to use the technology whom they acquire freely.

The form of re-elaboration of the license, discussed and agreed by the community, was a qualitative change with respect to previous versions, whose elaboration was basically an individual work.

The new version of the GPL does not annul the previous one or forces to release again the software under the new license version. And, although in principle they are incompatible (the GPLv2 does not allow to add any type of requirement that does not contain she herself), does not prevent in no way that programs with different versions from the GPL could be distributed together and even with their code fused under GPLv3 with its own requirements. In addition, there is much free software under GPLv2 license that it indicates that is applicable also to any later version of the license, which gives the option to also follow the terms of the GPLv3 if it is desired.

The GPLv3 introduces many changes, but it is more an evolution rather than a revolution and maintains the same objectives that the previous versions intact: to guarantee in the most effective possible form the four liberties of free software to developers and users. Because of that, it has been updated with the purpose of confront different technological and legal challenges that have appeared in the last years, very especially the programming patents and the systems of digital rights restriction (DRM).

The GPLv3 adds a greater protection by means of legal engineering to the patents threat: if any user tries to exert a right of patents against another one by means of the GPL, its license of use is finished. The GPLv3 also neutralizes the “tivoization” (the creation of a system that incorporates software under the terms of a copyleft software license, but uses hardware to prevent users from running modified versions of the software on that hardware), since it forces the manufacturer to provide a form in which the user can exert his right to the modification of integrated software GPL.

BSD

The permissive licenses, also known as BSD type, are a set of free licenses that are characterized by demand no condition at all, not even to maintain the same license at the time of redistributing a derived work. They are the licenses next to the public dominion and they are opposed to the robust licenses, or copyleft type, that only allows the redistribution of a derived work if the terms of the original license stay. Both types of licenses - permissive and robust they also consider free software; since they guarantee the four liberties (it frees use, copy, modification and redistribution).

The permissive licenses traditionally are known as BSD type, since all of them are based on the original license written at the University of California and was used in the Berkeley Software Distribution, a variant of UNIX. Although not as much as the GPL, the permissive licenses are also very popular and much free software is released under them. Examples of permissive licenses are the license of the MIT, the X11 and the licenses of diverse BSD projects (NetBSD, FreeBSD, OpenBSD). Examples of applications under permissive license are the X-Window system, Apache, Tcl/Tk, Ruby on Rails, the development platform MONO and parts of Mozilla.

These licenses allow the modification and the free redistribution of the source code or the binary form without no restriction, except for the recognition of the responsibility, something that by defect the own law demands in many jurisdictions.

For this reason, it is also possible to create and to redistribute versions derived as privative software from code under a permissive license.

Even the Free Software Foundation recognizes that, in certain cases, a permissive license can be strategically interesting to facilitate a standard or a free technology to be adopted: as it happened with the TCP/IP. That is also the case of the LGPL, that allows to use (to fuse or to connect) its code with software that is not free. Richard Stallman remembered years ago that the GPL (and copyleft) are not an aim in itself, but a device to guarantee the freedom of the users and who, in certain cases (like in the free format vorbis against the patented mp3) one permissive license, BSD style could be more beneficial for the freedom of the users rather than a copyleft type one.

Typical BSD license contains three clauses or sections, that allow the redistribution of the code (with or without modification), without no restriction and in any form (binary or source). The clauses simply make sure that the note of the copyright stays and that the registered marks and names are not used for non-predicted aims.

In addition, a warning of exemption of guarantee is included, thought to avoid legal problems with the software, and in some cases, the license has caused doubts on its validity in certain legislations that demand a certain guarantee when a certain service is offered. The interpretation of that clause in that case that the minimum guarantee that the law demands will be offered.

LGPL

The LGPL or GNU Lesser General Public License (previously known as GNU Library General Public License) was designed to establish a strategic commitment between the strict copyleft model (GPL type) - that forces to any connected or fused code to be compatible with the GPL and the model of permissive licenses (BSD type) - that allow to connect and to merge with privative code.

Version 3 was written up in the form of GPL linking exception, simplifying that form its content.

That is to say, now it is simply a GPLv3 license with some additional permission, most important of which it allows distributing works combined with the license that the user chooses.

LGPL contains the same restrictions that GPL license for the code that is released under it. Nevertheless, these restrictions are not applied to another program when this last one, a not-LGPL, is limited to compile or to connect, dynamically or statically, with the program under LGPL. Therefore, the main difference between the GPL and LGPL is that this last one can connect against (in the case of a library, to be used by) a program not-GPL, that can be free software or privative software. This type of licenses is classified as weak copyleft, against strict copyleft.

The idea of Free Software Foundation when they wrote down this license was that this way it would help the free libraries to be more popular (maximizing therefore the freedom of the users), when not imposing any restriction to the program that used them.

A characteristic of the LGPL is that it allows software under their terms to be turned into GPL software. This characteristic is useful if you want to reuse code from a LGPL library in GPL applications or libraries (for example, with the purpose of to prevent them to be used by privative software).

Mozilla Public License

This license has a great value because it was the instrument that used Netscape Communications Corp. in order to release its Netscape Communicator 4.0 and that way begin one of the most important project in the Free Software world: Mozilla. They have used it in great amount of free software products for daily use in all type of operating systems. The MPL is Free Software and promotes the collaboration effectively avoiding the "viral" effect of the GPL (if you use code licensed GPL, your final development must be under GPL). From a developer's point of view of the GPL presents a disadvantage in this point, and unfortunately much people close ranks to the use of this code. The MPL is not so excessively permissive as the licenses type BSD. These licenses are denominated of copyleft weak. The MPL (later the MPL) was the first new license after many years that take into account some points that were not considered by licenses BSD and GNU. It can be considered adjacent to the BSD license, but perfected.

Apache License

The Apache license (Apache License or Apache Software License for versions previous to 2.0) is a license of free software created by Apache Software Foundation (ASF). The license Apache (with versions 1.0, 1.1 and 2.0) requires the conservation of the warning of copyright and disclaimer, but the copyleft concept is not present on the license, since it allows to the use and distribution of the source code for free software and proprietary software.

Eclipse Public licence (EPL)

It is a free software license used by the Eclipse Foundation. It replaces Common Public License (CPL), eliminating certain terms referring to the patent. This license was designed in such way that it is friendly to the enterprise activity of free software, and owns characteristics of copyleft weaker than the GNU GPL. The receiver of programs licensed under EPL can use, modify, copy and distribute to the work and the modified versions, but is forced to publish its own changes. Nevertheless, if the receiver integrates a project licensed under EPL in a larger product, the object code can be relicensed under an arbitrary license, including a privative one. License EPL was approved by the initiative of Open Source (OSI) and Free Software Foundation (FSF). This license is incompatible with license GNU GPL. In 2017, the Eclipse Foundation published the EPL V2.0, which brings several improvements regarding to internationalization (suppress the reference to the "New York State law", add definitions of modified works and derivative works) and fixes the compatibility with the GNU GPL by allowing an optional "secondary license" that can be GPL-2.0+.

European Union Public License (EURL)

It was created and approved by the European Commission. It has a strong copyleft and it is consistent with the copyright laws of all the states of the European Union.

It is compatible and can be re -licensed explicitly to: GNU GPLv2, OSL 2.1 y 3.0, CPL 1.0, EPL 1.0, CeCILL 2.0 but it is not explicitly compatible with the GPLv3. It does protect against patents but doesn't care about DRM protection or tivoization.

It is translated to all the languages of the European Union.

It the license promoted by the European Union to be used in the works for the administration.

2.4.3 Compatibilities & Incompatibilities among licenses

After presenting the most used licence, we also like to present the incompatibilities among them. Based on the rights and duties that licenses give to both users and developers, these makes ones incompatible with others.

Table 2 shows the compatibilities and incompatibilities among the main open source licenses [Osami 2011].

Table 2: Compatibilities and incompatibilities between the main open source licenses

		I want to release a project under:						
		GPLv2	GPLv3	LGPLv2.1	LGPLv3	EPL	APACHE v2	EUPL
Modifies a code licensed under:	GPLv2	OK	NO [9]	OK if you convert to GPLv2 [1]	NO [3][10]	NO [3]	NO [3]	NO [5]
	GPLv3	NO [4]	OK	OK if you convert to GPLv3 [1]	OK if you convert to GPLv3 [1]	NO [3]	NO [3]	NO [5]
	LGPLv2.1	OK if you convert to GPLv2 [1]	OK if you convert to GPLv3 [1]	OK	OK if you convert to GPLv3 [1]	NO [3]	NO [3]	NO [5]
	LGPLv3	OK if you convert to GPLv3 [2] [10]	OK if you convert to GPLv3 [2]	OK if you convert to GPLv3 [2]	OK	NO [3]	NO [3]	NO [5]
	EPL	NO [7][12]	NO [7][12]	NO [7][12]	NO [7][12]	OK	NO [8]	NO [12]
	APACHE v2	NO [6]	OK [11]	NO [6]	OK [11]	OK	OK	OK [16]
	EUPL	OK [14]	NO [5]	OK [14]	NO [5]	OK [14]	NO [15]	OK
Links a code licensed under:	GPLv2	OK	NO	OK if you convert to GPLv2 [1]	NO	NO	NO [6]	NO [5]
	GPLv3	NO	OK	OK if you convert to GPLv3 [1]	OK if you convert to GPLv3 [2]	NO	OK	NO [5]
	LGPLv2.1	OK	OK	OK	OK	OK	OK	OK
	LGPLv3	NO	OK	OK	OK	OK	OK	OK
	EPL	OK [13]	OK [13]	OK [13]	OK [13]	OK	OK [13]	OK [13]
	APACHE v2	Upgrade the GPLv2 code to GPLv3 [6][11]	OK [11]	Upgrade the LGPLv2.1 code to LGPLv3 [6] [2][11]	OK [11]	OK	OK	OK [16]
	EUPL	OK [14]	OK [14]	OK [14]	OK [14]	OK [14]	NO [16]	OK
No	Arguments	References						
1	LGPLv2.1 gives you permission to relicense the code under any version of the GPL since GPLv2. If you can switch the LGPLed code in this case to using an appropriate version of the GPL instead (as noted in the table), you can make this combination.	http://www.gnu.org/licenses/gpl-faq.html						
2	LGPLv3 gives you permission to relicense the code under GPLv3. In these cases, you can combine the code if you convert the LGPLed code to GPLv3.	http://www.gnu.org/licenses/gpl-faq.html						
3	GPLv2 and GPLv3 only permits that modification to its code are covered by the same	http://www.gnu.org/licenses/gpl-faq.html						
4	GPLv3 is not compatible with GPLv2 by itself. However, most software released under GPLv2 allows you to use the terms of later versions of the GPL as well. When this is the case, you can use the code under GPLv3 to make the desired combination.	http://www.gnu.org/licenses/gpl-faq.html						
5	EUPL has a copyleft comparable to the GPL's. However, it allows recipients to distribute the work under the terms of other selected licenses, and some of those—the Mozilla Public License and the Common Public License in particular—only provide a weaker copyleft. Thus, developers can't rely on this license to provide a strong copyleft. The EUPL is incompatible with GPLv3, because recipients are not given permission to use GPLv3's terms, and the EUPL's copyleft conflicts with GPLv3's.	http://www.gnu.org/licenses/gpl-faq.html						
6	Apache v2 include certain patent termination and indemnification provisions that are not considered under GPLv2 and LGPLv2	http://www.gnu.org/licenses/gpl-faq.html						
7	EPL's weak copyleft and choice of law clause make it incompatible with the GNU GPL. EPL removes the broader patent retaliation language regarding patent infringement suits specifically against Contributors to the EPL'd program.	http://www.gnu.org/licenses/gpl-faq.html						
8	Incompatibilities with patents treatment	http://www.gnu.org/licenses/gpl-faq.html						
9	If you have the ability to release the project under GPLv2 or any later version, you can choose to release it under GPLv3 or any later version—and once you do that, you'll be able to incorporate the code released under GPLv3.	http://www.gnu.org/licenses/gpl-faq.html						
10	However, most software released under GPLv2 allows you to use the terms of later versions of the GPL as well. When this is the case, you can use the code under GPLv3 to make the desired combination.	http://www.gnu.org/licenses/gpl-faq.html						
11	Apache 2 software can be included in GPLv3 projects, because the GPLv3 license accepts Apache 2 software into GPLv3 works.	http://www.apache.org/licenses/GPL-compatibility.html						
12	Only the owner of software can decide whether and how to license it to others. Contributors to a Program licensed under the EPL understand that source code for the Program will be made available under the terms of the EPL. Unless you are the owner of the software or have received permission from the owner, you are not authorized to apply the terms of another license to the Program by including it in a program licensed under another Open	http://www.eclipse.org/legal/legalfaq.php						
13	The Eclipse Foundation interprets the term "derivative work" in a way that is consistent with the definition in the U.S. Copyright Act, as applicable to computer software. Therefore, linking to Eclipse code might or might not create a derivative work, depending on all of the	http://www.eclipse.org/legal/legalfaq.php						
14	If you substantially combine EUPL software with software distributed under a "compatible licence", the EUPL allows the resulting software to be distributed under the compatible licence instead of distributing it under the EUPL itself. The compatible licences are currently defined by the EUPL as being the: • GNU General Public licence (GNU GPL v.2) • Open Software licence (OSL) v. 2.1, v. 3.0 • Common Public licence v. 1.0 • Eclipse Public licence v. 1.0 • CeCILL v. 2.0	http://ec.europa.eu/idabc/servlets/Doc?id=32429						
15	Apache has a weak copyleft and it is not desirable for the purposes of the EUPL Community since it allows, to a certain extent, the creation of proprietary derivative works	http://ec.europa.eu/idabc/servlets/Doc?id=27472						
16	Software under several F/OSS licences can be combined with EUPL software. It is the case with all components that are licensed under licences that do not impose restrictions on future licensing (all the "permissive licences").	http://ec.europa.eu/idabc/servlets/Doc?id=32429						

2.5 Towards Business-Friendly open source ecosystems

Many observations show that it is possible to develop a business model with software licensed under any kind of open source licenses. RedHat is a good example of a company that generates a significant revenue from an offering built around software mainly licensed under Copyleft licenses. But it should be noticed that in the recent years, more and more open source projects use permissive or semi-permissive license as they are considered more "business-friendly". This is particularly the case for "platform" projects that are subject to be extended to create "products". A typical pattern is that the platform is made available under a permissive license (for example Apache or BSD), or a semi-permissive license like the Eclipse Public License, and some contributors create proprietary extensions that are marketed under a classical software vendor scheme.

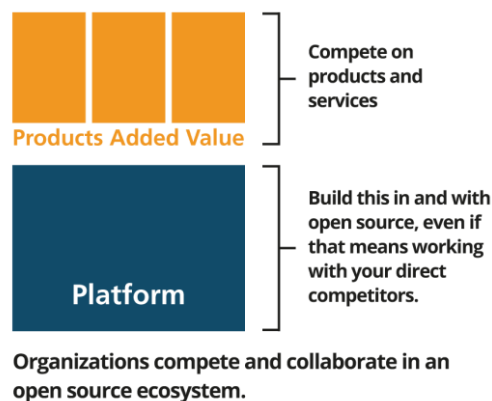


Figure 3: A business-friendly ecosystem based on extensible platforms

The previous figure shows this scheme as it has been applied with success in the Eclipse ecosystem since 2004. This resulted in thousands of products built on top of the Eclipse platform both by Software vendors and by Software integrators.

2.6 Issues of using Open Source Software in the Machinery Industry

Despite the high value proposed when using open source software, one also has to admit that there are some issues related to the use. Especially from the machinery industry, that does not sell software as a standalone, but may incorporate open source software in their hardware products, serious concerns have been voiced.

Due to the high number of different open source licenses existing that include different rights and obligations (e.g. information obligations) for the user, the risk of inadvertently breaching terms of licences should not be denied.

Typical license requests are "publish on each copy an appropriate copyright notice and disclaimer of warranty" "and give any other recipients of the Program a copy of this License along with the Program". In case of embedded devices and a multitude of open source software elements used, this is not an easy to fulfil requirement. In many cases, the effort for the manufacturer is tremendous.

Some standards for example request the user to provide the full source code with the software. In the usual case where the software has been modified to fulfil the application's requirements, the manufacturer may see their process knowledge under threat when having to publish the complete source code.

In many cases, the license text must be part of the contract – thus you would need to give access

to the software before signature of the contract. In other cases, license text need to be included in the (embedded) device.

A significant number of cases have caused legal action and have been taken to court for copyright infringement, especially with GPL-2.0 licences. This can imply very high costs for the manufacturer (Injunctive relief, claim for damages, annihilation, recall from the supply chain, claim on reimbursement etc.) and lead to a complete stop of sales for the product concerned (i.e. a machine sold with the respective open source software).

To overcome those hurdles for the user in the machinery industry who is not so familiar with the use of open source software, the licence models chosen within the project should be kept simple, to a low number, and favour the use of business-friendly open source licenses. Maybe also a short guideline could be produced to make usage of software provided as easy and beneficial as possible for the user.

3 Preliminary Market Analysis

3.1 Market Size and Share

3.1.1 Market size in robotic-related domains

Robotics is an enabling technology both for the manufacturing industry as well as a growing number of service sectors. Overall, a robotics-related increase of the EU GDP by EUR 80 billion by 2020 is expected by the robotics community in their strategy research agenda [SPARC 2013].

Robotics technology can be deployed in a wide range of different market domains. Each domain has its own needs and requirements. The Multi-Annual Roadmap for Robotics 2020 [Roadmap 2015] provides an overview on the technology trends and market opportunities for the most important robotics domains.

The by far largest and most important application domain today is industrial robotics – robots deployed in the manufacturing environment. With \$13.1B of revenues in 2016, industrial robots contributed to 61% of the global overall robotics market [IFR-WorldRobotics 2017]. Largest industries applying industrial robots are the automotive and electronics industry. Main driver for global growth is the electronics industry.

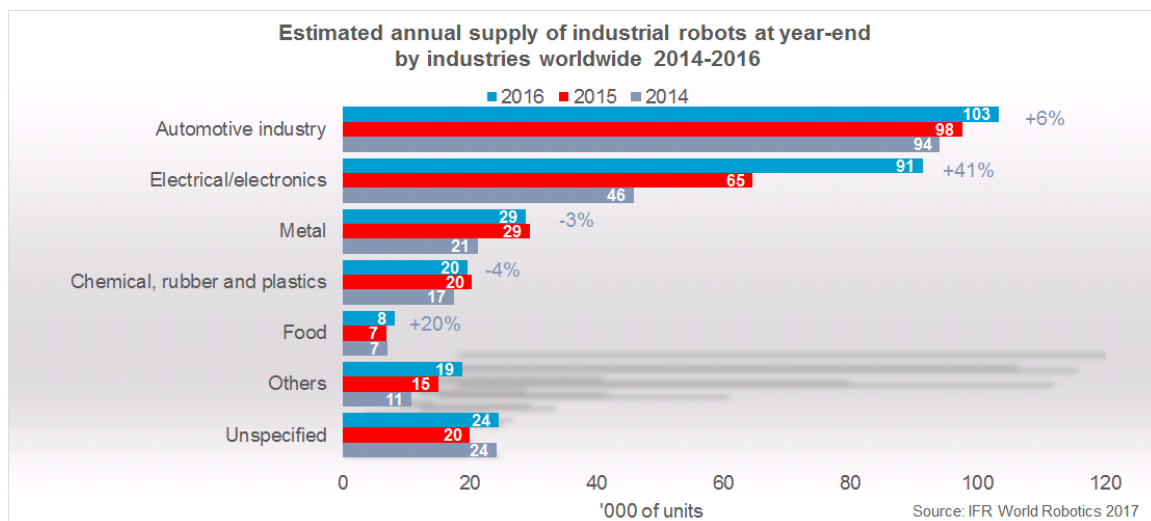


Figure 3: Main application industries of industrial robots [IFR-WorldRobotics 2017]

Professional service robotics make up about 22% of today's global robotics revenues. Main application areas are medical, logistics, field and defence. The costs per system vary drastically for the different application areas -from ~\$16,000 on average for a public relations robot to \$1M for a medical system.

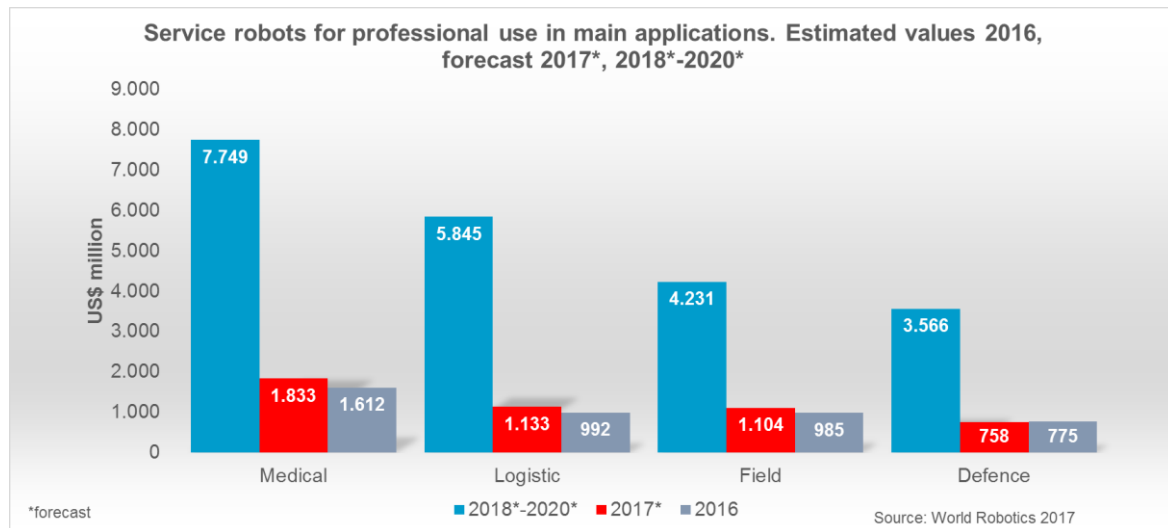


Figure 4: Main application sectors for professional service robotics [IFR, WorldRobotics 2017]

Robots for personal and domestic applications and entertainment make up the third group. Unit prices here are even less, they already have become commodities.

The robotics market is complex involving a diverse range of opportunities. Organisations may create value by concentrating on specific end applications, supplying different types of robot, modules, sub-systems, tools, or providing services within the market. It also includes dedicated supply chains, design services, and research and development organisations. Providing a coherent categorisation of the potential in each type of market is an important step in evaluating the potential for robotics and robotics technology.

3.1.2 Market size in robotic software and services

The robotics hardware market will grow from \$21.4B (according to the IFR, the International Federation of Robotics) in 2016 to \$40.1B in 2020, growing at an average compound annual growth rate of roughly 15% [Murphy 2017]. This estimate only includes hardware sales, not software or other supporting services for robotics. When factoring in these additional markets, it is estimated that the total robotics market value could be 3x larger. It is anticipated that hardware and components used for robotics will largely be commoditized over the next 10 years. Differentiated hardware will be the exception to the rule and unique value will be driven by software and accompanying services.

Not all robots are deployed once and then used over their complete lifetime. This leads to the development of new business models, away from classical sales to lease or rent models ("Robots as a service", RaaS) or even the complete service including operators is rented (e.g. for maintenance and inspection tasks).

The report on robotic software platforms market [Technavio 2016] estimates that this will reach USD 8.15 billion by 2020, should the market sustain a CAGR of 4.57% during the forecast period.

3.1.3 Market size on software and modelling tools

Analysts' reports, as the VDC report [Girard 2016], project that the global software and system modelling tool (SSMT) market will expand from \$916.3M in 2014 to reach \$1228.8M in 2018, a

Compound Annual Growth Rate (CAGR) of 7.6%. Increasing system complexity is the primary driver of modelling tool revenue growth in both the embedded and enterprise/IT market segments. The volume of new users is expanding as any organizations find modelling tools are an effective approach for overcoming the challenges their software developers or engineers face designing increasingly complex software and systems. Extensive process standards and regulatory requirements will continue to be leading drivers of modelling tool use in the embedded market, especially in safety-critical industries. The influence of mandates on development tool use is also rising in the enterprise market as financial reporting requirements become more common.

3.1.4 Market size on robotic software platforms

The demand for robotic software platforms is scaling up as they enable industries to reprogram the machines frequently, modify, and test their robots to suit multiple application demands. In 2015, the APAC region dominated the global robotic software platforms market with a 66% stake in the overall market share. The market is booming in this region as many manufacturers are adopting robots in their plants to increase productivity and meet the demands put forth by industries such as automotive and healthcare. Government orders and offers that encourage local robotics industry to boost exports, loans at low interest and other incentives for robotic factories, and rising labor wages are some of the factors that will contribute to growth in the market in this region in the coming years.

3.2 Market Trends

There are a number of market trends that confirm the market opportunities for the RobMoSys tool-chain(s), which will be valuable not only in robotics but in all markets where embedded systems are willing to incorporate autonomous capabilities.

3.2.1 Time to market & competitiveness

In an increasingly competitive market, with an ever-increasing speed of innovation, organisations must pursue easier, faster, and more efficient ways to build and assure robotic software systems to remain competitive. The cost of adopting a robotics system varies depending on the application and software account. Software platforms, such as RobotStudio offered by ABB or iRobot Aware by iRobot, can be used to program or test robotic automation systems. Compared to the traditional teach pendant method that consume 4-40 hours, these software platforms allow offline programming, and therefore are more feasible and reduce production downtimes. The offline program is coded on a personal computer using C, C++, and Python or URBI, depending on the vendor's preference.

3.2.2 Reuse and cross-domain harmonization

The trend and need is to increase reuse-geared development and integration processes as a major means to reduce costs. The challenge with reuse is not necessarily with the reuse itself but with reuse of composable artefacts, especially where artefacts are cross-domain. A robotics software artefact for manufacturing robots might not be able to be applied directly “as is” in the healthcare or agriculture domains for example.

The most important goal of any robotics platform company is to see their product being pre-installed on a mass-produced commercial robot (just like Windows is pre-installed on new PCs). This is still yet to happen.

The RobMoSys approach will also help robotics technologies to be accepted in safety-critical domains (automotive, railways, etc.) and applications, and would prepare robot technologies to certification. Robotic software, by nature, inherits the requirements of the system in which it should

work. When “immersed” in a specific domain, the development practices of the hosting domain will be immediately imposed to the robotic one. After some periods of disorientation arguing that what is done in safety critical domain to validate software cannot be done to validate robotic software, standardisation bodies will eventually put new requirements on methods for robotics software validation.

The risk that robotic technologies will remain only in particular niche markets, e.g. autonomous shuttles for private sites, or in established ones, such as industrial automation but without concrete perspectives of taking advantage of their real innovation potential (e.g. autonomous capabilities), is directly related to the capacity of robotics to embrace mature software and validation methodologies, as proposed by RobMoSys. Here, the availability of more formal models is expected to be the key differentiator.

3.2.3 Open solutions

The use of industrial robots is common in the automotive, metal, electronic, and healthcare industries. This rise is attributed to the technological advances and user friendly and simple programming software provided by third-party vendors and OEMs. The software platform and the training sessions, required before installing robots in industries, are expensive. However, the ease of offline programming, reduction in production and equipment downtime, and the profit earned have expanded the use of robots in these industries. The availability of open source software from third party vendors (such as OROCOS and OpenJAUS) that are compatible with robots of different makes are motivating end-users to use machines in complex manufacturing processes.

Increased adoption of software modelling tools, however, has not translated into equivalent increases in commercial market revenue to the degree that VDC report [Girard 2016] had previously anticipated. VDC suspects that a number of factors, including the availability of low-priced and open-source modelling tools, have significantly impacted the growth of the commercial market. Recent business trends are also encouraging proprietary language-based modelling tools move to common open-source platforms backed by private funding to allow tools to rapidly adapt to customer needs [Eclipse 2016]. RobMoSys objectives are completely in line with these trends.

On the industrial robotics side, most established manufacturers of industrial robots provide their own platforms for programming their robots (e.g. look at ABB’s robotics software products). They would probably prefer selling their own products other than using third-party products. To counter this tendency, OMG robotics group has recently staged an orchestrated campaign to create a set of standards which would break such vendor lock-in. The efforts are in the very early phase, but we hope they will produce working standards in medium-to-short term. A similar but independent development is going on in Europe – look at European Robotics Platform web site.

The revenue model of the software market is diverse and vendors compete on the basis of suitability for their business. The most preferred revenue model of the robot manufacturers is the hybrid model, wherein end-users are charged a fixed amount for software licensing, support and services, and management. But the easy availability of open source software from vendors such as OROCOS and Gazebo is encouraging end-users to reprogram and simulate robots in the plant. An open source software provider only charges for support and services opted by end-users

3.2.4 Increasing service life with shortening product lifecycles

The average service life of an industrial robot is estimated between 12 to 15 years [IFR, WolrdRobotics 2017]. That does not imply that robots are used in the same application, e.g. manufacturing process over the whole time. Considering the shorter product lifecycles, we see, for example in the electronics industry, that setups need to be reconfigured several times during the service life of the robot.

Given the rate at which technology is developing, tools, methods and documentation that are used for the original product need to be available throughout the product lifespan. This is so that incremental development and bug-fixes, for example, can be performed cost effectively. Moreover, the ease of re-usability and re-programmability becomes increasingly important to reduce the costs for resetting the production cell or service robot application.

3.2.5 Modularity

Advances in technology have transformed robotics from handheld/static instruments to remote technology-enabled machines and from teaching to self-learning entities. One such innovation is the self-configurable modular robot, which is created by connecting a number of modules with memory units and CPUs. These machines can change shapes depending on the task at hand. Each module of such new generation robots can communicate with the others and take decisions before performing a task.

Two of the best concepts in the robotics field that have emerged recently are modular and transformer robots. The transformer robot uses the same structure to make various forms. Such machines can change or transform to two different shapes even though one central unit controls all components.

Thanks to the focus on composability and compositionality, as RobMoSys does, a growing number of companies will be able to contribute software modules that can become commodities for the ecosystem, drastically lowering the entry barrier. This situation will only be reached if the underlying models are sufficiently fine-grained to allow the clear separation between “commodity” and “business sensitive” modules and models.

3.3 Stakeholders and their Needs

The know-how acquired in RobMoSys is expected to be strategic for European organisations. The RobMoSys stakeholders are shown in Table 3, and concern those actors who will be directly, or indirectly, positively affected by the RobMoSys ecosystem, its activities, and/or its results.

Table 3: RobMoSys Stakeholders

Target Group	Examples of stakeholders
Industry: transport and logistics, manufacturing, healthcare, agriculture, and other possible domains that RobMoSys could have an impact.	OEMs, component suppliers, integrators of robotics systems, tool vendors, consulting and service providers, certification organizations, standardization groups and industrial forums. SMEs constitute a special interest group for RobMoSys, as they generally have very limited access to basic or applied research to develop new products.
Policy makers	Consultancy providers, assessor companies, standardization and regulation bodies.
Research community	Universities, research institutes.
Scientific communities	Robotic software development, quality assurance communities.
Open-source communities	Developers of open-source tools for CPS.
Users of Robotics Systems	People that will ultimately use robotics systems or their services.

3.3.1 Industry

The industry represents the key target group and its stakeholders are in the main scope of the

project's dissemination activities. However, it is important to recognise that the industrial community consists of heterogeneous groups that often operate in relative isolation, and they need to be targeted individually. In particular, different communities of "practices" are likely to be interested in different aspects of RobMoSys. This depends on their role in the supply chain, activities during development, assurance and certification of the products on the one hand, and details of the challenges currently faced by individual industrial sectors on the other. These varying interests will influence the RobMoSys business models, and the consortium will ensure that the content of different business models is appropriate for the target audience.

Original Equipment Manufacturers (OEMs)

In applications domains such as manufacturing, OEM refers to the manufacturer of the original equipment, that is, the parts assembled and installed during the construction of a new manufactured product.

A more recent definition of OEM is a company that buys a product and incorporates or re-brands it into a new product under its own name. If the OEM is a robot manufacturer it represents the highest level of integrators, who needs support from their suppliers to comply with the assurance and certification process for sub-systems within a robot.

Component Suppliers

Within the supply chain of components, key component suppliers are responsible for assuring the critical properties of their delivered products. Component suppliers need to support higher level integrators in their assurance and certification processes.

Integrators of Robotics Platforms

Within the industrial setting, platform integrators are ultimately responsible for the dependability (e.g. safety) of the products delivered to the end users of the consumer market. These integrators are referred to as OEMs in some domains, e.g. transportation systems. They typically take primary responsibility for the assurance of the platform (e.g. assessment of safety), integration of the overall sub-systems, and certification.

Consulting and Service Providers

Consulting and service providers support OEMs, component suppliers, and integrators of robotics platforms during the assurance process. They perform testing and prepare assessment documents. They also make reviews and perform verification activities, e.g. inspections.

Certification Organizations

Certification organizations support OEMs, component suppliers, and integrators of robotics platforms regarding assessment during the assurance lifecycle. They are also acting as assessor companies to perform independent assessments.

Tool Vendors

Software tool vendors (organizations specialized in making and selling software tools) support both platform integrators and component suppliers, and they facilitate the exchange of relevant information between all supply chain and RobMoSys stakeholders. Consequently, support and buy-in from tool vendors is critical for the medium- to long-term success of the development tools, as developed in RobMoSys. Ultimately, the vendors buy-in has two aspects. Firstly, the project must promote the adaptation of the existing tools to the RobMoSys architecture and working philosophy. Secondly, it is necessary to generate sufficient interest in a tool vendor community for supporting the framework and, thus, to promote development of tools that will implement and enhance the RobMoSys architecture and (prototype) tools.

3.3.2 Policy Makers and Standardisation Groups

Policy makers represent stakeholders for standardisation and regulatory bodies. The development, analysis, and assurance of robotics systems is regulated and standardised in most domains. Standardisation and regulatory bodies are key stakeholders in the assurance and certification process and, consequently, they represent a key stakeholder that must be targeted by RobMoSys business models.

Standardisation for robots is usually done on international level handled by the International Organization for Standardization (ISO). This includes both safety standards and definitions, as well as standards on performance criteria and test methods or criteria for the modularity of systems.

Before placing a robot on the market in Europe, the installation needs a CE marking / declaration of conformity with the machinery directive. Application of the specifications defined in the respective robotics safety standards on the basis of a risk assessment performed confers a presumption of conformity with the essential health and safety requirements of the Machinery Directive. For this reason, the safety standards are harmonized on European level by the European Committee for Standardization (CEN). The machinery directive is converted into national law in each member state, Making both the European Commission (DG Grow) and the different national ministries relevant stakeholders.

Also organisations certifying the safety of system, like TÜV in Germany, are relevant stakeholders in the context of RobMoSys.

3.3.3 Scientific and Research Communities

RObMoSys regards the exploitation of project results to the scientific and research communities as one of the key aspects of the overall business models. It will allow to extend the impact of the project by galvanising research community and, potentially, reaching out the industrial contacts of other research organisations that are not currently covered by RobMoSys partners' contact networks.

3.3.4 Open Source Communities

RobMoSys aims at supporting the open source philosophy, which promotes free redistribution and access to an end product's design and implementation details. Opening the source code enables a self-enhancing diversity of production models, communication paths, and interactive communities. The open-source software movement has created a new environment for which new copyright, licensing, domain, and consumer issues were created.

The open-source model includes the concept of concurrent and different agendas and various approaches in production, in contrast with the closed source and centralized models of development such as those typically used in commercial software companies. A main principle and practice of open-source software development is peer production by collaboration, with the end product, source-material, "blueprints", and documentation available at no cost to the public. The open source communities form a very generic group of participants, from companies such as IBM to university-based contributors. It is rather seen as an exploitation strategy than a stakeholder representative. For RobMoSys a specific number of open source communities is relevant and can be seen as special stakeholders: Eclipse, Polarsys, OROCOS, ROS, Gazebo.

3.3.5 Users of Robotics Systems

The users of robotics systems are the people that will ultimately use the robotic systems and their services. They are the people who will work most closely with OEMs and integrators to generate results. They are interesting and valuable for RobMoSys from the robotic system operation and maintenance process viewpoint.

These users can help identify more real dissatisfaction than almost anyone else in the robotic system production and operation chain. RobMoSys should approach them to understand their needs and what the RobMoSys ecosystem needs to do for them in order to get results.

RobMoSys must create value for them by helping them overcome the constraints (e.g., costs, resources, maintenance, reuse).

Table 4 provides a summary of needs per stakeholder type.

Table 4. Needs of RobMoSys Stakeholders

Stakeholders	Interested in:
Original Equipment Manufacturers (OEMs)	<ul style="list-style-type: none"> Complying with the standards for development, assurance and certification. Enabling an efficient tool supported development process which suites the needs for robotic software and documentation in an optimal way. Organizing the suppliers work related to the robotics products to achieve efficient supplier coordination.
Component Suppliers (Manufacturers)	<ul style="list-style-type: none"> Specification of robotics modules, which can be integrated into the overarching robotic systems. Transferring component artefacts (e.g. specifications, assurance results and documentation) across multiple domains. Preserving the integrity of the components that they provide to platform integrators. Ensuring the integrity of the components both up- and down-stream of the supply chain. Application of tool support for the provision of composition of qualities and other properties so that integrated systems can be created with lower effort.
Integrators of Robotics Platforms	<ul style="list-style-type: none"> RobMoSys results concerning the composition of the robotic software components based on individual modules, and that ensure the integrity of the non-functional properties passed through the supply chain. Tools that support these processes.
Consulting and Service Providers	<ul style="list-style-type: none"> RobMoSys results that ensure the integrity of the components passed through the supply chain.
Certification Organizations	<ul style="list-style-type: none"> RobMoSys results concerning intra/cross-domain and multi-concern assurance.
Tool Vendors	<ul style="list-style-type: none"> Interoperability. The adaptation of the existing tools to the RobMoSys architecture and working philosophy. Information access. Ensure that all the information relevant for the tool development is available.
Policy Makers and Standardisation Groups	<ul style="list-style-type: none"> Assurance and certification process.

Scientific and Research Communities	<ul style="list-style-type: none"> • Outcomes of the project. • Advance of the state of the art.
Open Source Communities	<ul style="list-style-type: none"> • An open tool platform developed by collaboration and with free redistribution and access to an end product design and implementation details.
Users of Robotics Systems	<ul style="list-style-type: none"> • Meet their requirements and constraints in terms of operation and maintenance.

3.4 Opportunities and Risks

RobMoSys strength comes from the strategy to secure adoption in the robotics community and sustainability and from the authoritativeness and complementarities of project partners, representing a guarantee of reliability and credibility. However, it cannot be ignored that the RobMoSys business success presents some risks. These are listed below, and they are shared with several partners' own business plans.

3.4.1 Unavailability of success stories

For new tools or approaches, in order to become affirmed or de-facto recognized, all domains of interest (manufacturing, agriculture, transport and logistics, etc.), are characterized by long latency. This might prevent applying the RobMoSys results to their full extent in industrial use cases before the end of the project. Therefore, here is a risk that RobMoSys remains a valuable research project only. Essential will be to establish and determine pilot customers or users, at least, and support them early on. The methodology will be demonstrated through pilot cases developed by third-party to further involve people external to the project, and get useful feedback. Last but not least, intensive dissemination activities are planned.

3.4.2 Inadequacy of use cases and requirements

Use cases are essential to demonstrate the business value of RobMoSys approach and tools. That means the use cases must meet business demands and shall not be artificial. Equally requirements that are derived from the identified business needs. It will be essential to have periodic reviews and collection of feedback. The strategy of RobMoSys is entirely devoted to secure adoption in the robotics community and sustainability. The situation today is that the robotics community has not developed yet common methods for software development, so that adoption of the RobMoSys approach could be an issue, mainly for developers. For this reason, RobMoSys has been built around a simple but effective concept: the involvement of the robotics and the software community. To that end, all the project partners' activities are massively focused on preparing appropriate specifications (concerted with Tier-1 experts) and running Open Calls.

3.4.3 Generic or complex results

With the ambition of meeting in one shot all robotics domains, the RobMoSys achievements might result too generic and complex at the same time, and thus scarcely acceptable by any single domain representative. Continuous advice by RobMoSys industrial partners is essential to mitigate this risk. RobMoSys intends to create as well synergies with other robotics platforms, as ROS and I-Cub, to further improve them. As for software, selection criteria in Open Calls will secure an open-source software and tools "track" for the eco-system and commoditization of basic building blocks of certifiable quality.

RobMoSys proposes a technical solution based on a **model-driven approach** that can be tailored to stakeholder specific needs, but practitioners could refuse more formal, not code-based

approaches. Special information events, targeted campaigns and training will be undertaken. The project aims at showing the benefits of the approach since the very beginning through trials during the Tier-1 workshops especially targeted to software developers. Feedback will be taken into account to remove blocking points. RobMoSys will create related software projects in different platforms, such as Eclipse, and web-based repositories, such as GitHub.

3.4.4 Difficult integration

The RobMoSys platform might result difficult to integrate with available tools and practices, unless spending time to import their know-how within this new platform and reducing its probabilities to be adopted. Periodic integration trials should be attempted to mitigate this. During the preparation of the specifications, to avoid imposing a quality standard unilaterally, representatives of industry and academia will be highly involved in the preparation of the smallest possible set of common concepts to share in the community as renowned best practices and methods. All the specifications will be publicly available, and the project's team activities constantly monitored by the community.

3.4.5 Visibility

RobMosys's visibility may be too low. RobMoSys proposes different instruments (Tier- 1, Open-Calls, RiF and CoC), that could cause some dispersion and loss of efficiency in the global communication. A clear communication plan including presentations at broad-spectrum and specific events will likely resolve this problem – just as some RobMoSys partners did very successfully within ECHORD and ECHORD++. Outreach to new potential communities is also secured by specific actions carried out by two foundations that hopefully will have a multiplier effect in their respective already established communities. The first list of members gathered for the Tier-1 group already witnesses the interest that the different communities may have in the project. Platforms and facilities will be chosen depending on the requirements and recommendations gathered during the project lifetime.

4 Preliminary RobMoSys Business Models

4.1 Overall Strategy for RobMoSys Business Models

The goal of this document, as a first iteration of RobMoSys business models, is **establishing the basics and on specifying the available knowledge of markets, stakeholder needs and best business approaches** for the RobMoSys partners. We do not establish the definitive business approach for RobMoSys, but a set of alternative schemes that can fit RobMoSys requirements. We plan to use this baseline to discuss with relevant RobMoSys stakeholders and experts during the workshops to be held after the first year of the project. We also will elaborate questionnaires of semi-structured open questions to get feedback from these actors.

During this first year of project, the focus of the business models is on defining general approaches for business development based on the current market trends and RobMoSys opportunities. We split the business models in three areas:

1. Community Business Models, mainly related to funding the RobMoSys community activities.
2. Product Business Models. In this first iteration of the Business Models for RobMoSys, we use **only the Business Canvas formalism** to specify the potential business models for RobMoSys.
3. Business Models per Partner centred on gathering their views for the licenses schemes.

4.2 Community Business Models

This section is focused on external finance of the RobMoSys community activities and future projects. In these cases, there is some type of external flow of capital that is in charge to contribute resources for the community development. This way, the developed free software can be considered, in some way, a product of this external financing. For that reason it is common that this external source decides (at least partially) how and in what the resources are spent.

In a certain sense, this external financing for free projects can be considered like a type of sponsorship, although this sponsor does not have to be disinterested (and usually it is not).

4.2.1 Public financing

This is a very special type of financing of free projects. The financial organization can directly be a government (local, regional, national or even supranational) or a public institution (for example, a foundation). In these cases, the financing usually is similar to the one of the projects of investigation and development, and in fact it is very usual that it comes from public organizations that promotion R+D.

Normally, the financial organization does not look for recovering the investment (or at least not in a direct form), although it usually has clear objectives (to help creating the industrial and researching ecosystem creation, to promote certain technologies or certain type of applications, etc.).

Some of these projects have as part of their objectives, to improve and create free software in their research area (like a tool for the research, or product derived from it). The motivations for this type of financing are very varied, but we can stand out the following:

- **Scientist.** This is the most usual case in projects of investigation financed with public sources. Although its objective is not to produce software, but researching in a certain field (related or not to computer science), is very possible that for achieving in that research they need to develop programs that are used as necessary tools. Normally the project is not interested in commercializing these tools, or it is even interested in other groups to use

them and improve them. In these cases, it is quite usual to distribute them as free software. In this case, the resources the group used for the research were partly dedicated to the production of this software, that's why it is possible to be said that it was developed with public financing.

- **Promotion of standards.** To have a reference implementation is one of the best ways to promote a standard. In many cases that supposes to have programs that are part of that implementation (or if the standard talks about the software field, that is the implementation they themselves). To become a reference implementation to become useful for the promotion of the standard, it is necessary that the code is available, to at least verify its interoperability, for anyone who wants to develop products for that standard. And in many cases, it is also convenient that robotics manufacturers can directly adapt the reference implementation to use it in their products, if they wish so. Usually, whoever is in charge of the promotion is a public organization (although sometimes it is a private partnership).
- **Social.** Free software is a tool of great interest in the creation of the basic infrastructure for the information society. The organizations interested in using free software to help the society of the information universal access, can usually finance projects related to free software (projects for development new applications or adaptation the existing ones).

4.2.2 Private non-profit financing

This it is a type of financing with many characteristics similar to public finance, where the financing is normally done by non-governmental foundations or organizations.

The direct motivation in these cases is usually to produce free software for its use in some specific field that the financial organization considers especially relevant, but also can be the indirect motivation to contribute to solve a problem (for example, a foundation dedicated to promote the research on a disease can finance the construction of a statistical program that helps the analysis of experimentation groups where that disease is studying).

Generally, not only the reasons to realise this financing but their mechanisms are very similar to those of public financing, although naturally they are always marked by the objectives of the organization that finances.

Probably the main example of a foundation that promotes the development of free software is Free Software Foundation (FSF). Since mid the decade of 1980 this foundation is dedicated to the promotion of project GNU, and to promote the development of free software in general.

4.2.3 Financing by whoever needs improvements

Another type of financing for the development of free software, no longer so altruistic, is the one that takes place when somebody needs improvements in a free product. For example, a company, for internal use, needs certain functionality in a given program. Or it needs that certain errors in a program are corrected. In this type of cases, the usual thing is that the company at issue contracts the development that needs. This development, very usually (or because it imposes the license of the modified program, or because the company decide so) is free software.

4.2.4 Financing with related benefits

In this case, the financial organization is looking for benefits by products related to the program whose development is contributing with resources. Normally, in these cases the benefits that the financial company obtains are not exclusive, since others can also enter the market of the related product sale. But either the quota of market that it has is sufficient enough to share it with others or has some clear competitive advantage.

Some examples of product related to software are:

- **Books.** The company sells manuals, user's guides, texts for courses, etc. related to the free program that helps to finance. By all means other companies can also sell related books, but normally financing the project will give the company access to key developer before to its competitors, or simply assures this way a good image of the company facing the users' community.
- **Hardware.** If a company finances the development of free systems for a certain type of hardware, then the company can sell that hardware more easily. Again, as the developed software is free, it can appear another competitor that sells devices of the same type, using those developments, but without collaborating to the project financing. But even then, the initial company has several advantages from its competitors, and one of them can be that as it is in charge of the resources for the project, that position allows the company to influence in the developments the company is more interested to increase their development priority.
- **CDs with programs.** Probably the more popular model of this type is the one used by companies that finance certain developments that apply to their software distribution. For example, having a nice desktop environment can help much to sell CDs with a certain distribution of GNU/Linux, and therefore, to finance its development can be a good business for sells those distributions.

It is important to realize that, this type of financing has the intention of profit, so the financial organization has to perceive some possible benefit in this financing. In real cases, however, it is usual to have a mixture between a profit and altruism spirit.

Just to mention an example, some big end-users interested in advanced robotic systems, such as: Procter & Gamble, Airbus, the car manufacturers and other large automation-savvy manufacturers such as VDL, supermarket with warehouse backends such as Ocado, Carrefour, Ahold, Lidl, etc. Could finance the development of robotics platforms. These companies could receive some benefits for the rights of RobMoSys assets. These companies could also collaborate to assure the RobMoSys community's continuity, which may be especially critical for companies interested on open source software, or their clients, when the company's main business is to sell equipment with this software pre-installed.

4.2.5 Financing as internal investment

There are companies that, directly as part of their business model, develop free software. For example, a company can decide to initiate a new free project in an area where they perceives as a business opportunity, with the idea to make profitable their investment later. This model could be considered as a variant of the previous one (indirect financing), being the "related benefits" the advantages that the company gets on the free program production.

This type of financing makes possible several business models. Perhaps in some cases, the software is developed just to satisfy the needs of the own company, and later on the company decides to release it, and perhaps open a business line related to it.

For example, small companies, the innovative system integrators like Intermodalics, Smart Robotics, etc. might want to support the modelling improvements, since that allows them to reuse their own effective implementations in more "standardized" applications.

4.2.6 Other financing models

There are other financing models hard to classify among the previous ones. We can select these:

- **Use of markets to bring developer and clients into contact with one another.** The idea that sustains this type of financing is that, mainly for small developments, it is difficult that a

client could contact a developer that can undertake the client needs. In order to improve this situation, the free software development markets appear where the developers would publish their abilities and the clients the developments they need. When a developer and a client agreed, we have a situation similar to the already described as “financing by whoever needs improvements”.

- **Stocks sales to finance a project.** This idea of financing is similar to the stocks exchange o, but oriented to the development of free software. It has a few variants, but one of the most popular works as it follows. When a developer (individual or company) has an idea of a new program or an improvement to an existing one it writes, in a specification form, and what he considers the cost that would have this development, and emits stocks for the implementation. These stocks have a value that is only executed if the project is finally finished. When the developer has sold sufficient stocks, the development begins, that is financing with loans based on the stocks. When the development finishes a third independent part certifies that it fulfils the specifications, the developer “executes” the stocks that had sold. The users, who wished that new program, or that improvement to one already existing one, would be the interested clients that buy the stocks. Somehow, this stocks system would allow that the clients could determine (at least partially) the priorities of the developers. This would also allow that other organization different from the one that has the idea could assume the development costs.
- **Developers’ cooperatives.** In this case, the free software developers, instead of working individually or for a company, can meet in some type of association (normally similar to a cooperative). Their operational is similar to a company, with characteristic ethical commitment with the free software that can be part of their statutes. In this type of organizations could appear mixtures of voluntary work with remunerated work. An example of this type of organization is Free Developers.

4.3 Product Business Models

We use business model concepts based on the Business Canvas structure. These business models have been defined for Polarsys [OPEES 2013]. In this section, we evaluate its suitability for the RobMoSys ecosystem.

The RobMoSys consortium will not only reuse OSS, but is also open sourcing its own code to make it sustainable beyond the end of the research project. By experience, when code is not open sourced, it is really difficult to keep enough interest from a large part of consortium members.

The RobMoSys consortium will be able to take advantage of the open source on the following business models.

4.3.1 Pure Services

In this model, a service company builds expertise around a number of free software works and helps its clients with installation, support, use, maintenance, upgrades and adjustments or customizations that may be needed. This business model is based on a relative abundance of software, but scarcity of expertise, to market service. An example is Alcôve, a European company providing professional free software services. Another interesting example in this domain is what IBM is doing with Linux with over 2,000 Linux-skilled IBM Global Services professionals. Other companies very actively promote Free software services, like ATOS Origin.

Among the pure service players, two different kinds of services can be distinguished:

- (1) Tool services: In this situation, the pure service player provides services associated with the tools themselves. For instance, it offers to integrate such tools into an existing development infrastructure, or to train staff using such tool, etc. In some circumstances it may partner

with an infrastructure provider to make a comprehensive offer, or even act as an infrastructure provider.

- (2) Application services: In this situation, the pure service player provides services related to the final customer application. For instance, in the context of Polarsys, it is the position of a pure service provider offering to develop part of the embedded software.

However, in real business life, pure service players are used to offer bundles covering both aspects. In this case, the pure player becomes, first, a referrer of the tool technologies, then a developer of such technology if any adaptation is needed, and thirdly a user of the resulting technology during the project.

Offering service, and in this case expertise, is certainly interesting in the context of dependable systems development, if the consultants being engaged are experts and contributors to the free software being used in the construction of the dependable system. The difference between this business model and that of “regular” service companies is that in the free software case consultants can inspect and contribute to the development of the free software product. As such they are in a position to provide a deep level of know-how to the client developing a dependable system. By fostering technology reuse, they will also be able to provide lower price than their proprietary competitors.

The Pure Service business model can be summarized as displayed below in Osterwalder’s canvas:

Key partners	Key activities	Value proposition	Customer relationship	Customer segments
Free software communities that design the products used in customer projects	Integration of free software and redistribution back to community of the most important changes/fixes/enhancements.	Outsourced integration service	Tight, or even symbiotic	Large corporations
	Key resources		Channels	
	Software developers for technologies used		Direct contact, with colocated teams	
Cost structure		Revenue streams		
Mostly time-based		Either call for tender (specific contracts at high prices) or maintenance contracts (subscription-like).		

Pure service providers have a critical role in the RobMoSys ecosystem because they are the most significant technology providers, at least in terms of available human resources.

Customer Segment

The pure service players are targeted on customers segments that need high-value services and can afford to pay for. The typical end-user members of the RobMoSys ecosystem are part of their natural customer segment: in the process of designing robotics systems, they need complementary resources or expertise; the projects they are involved in are million-euro project for which failure cannot be considered. Therefore, they are ready to pay for services that help carrying them out.

Value-Proposition

From a technical standpoint, the offer of pure service players can be summarized as follows: they offer to the customer a global and comprehensive expertise that is used to implement a specific project. The customer can have several reasons to use a service provider. First, it may not have internally the skills required to carry out such work, or not have them in sufficient quantity or

availability. Second, it may want to avoid using internal resources for activities that are not at the heart of its business. Thirdly, it may have financial constraints that push in favour of using a service provider: the work is done through an external service contract, not by employees, so it has a number of consequences in terms of accounting, finances and labour law. Therefore the real value proposition of service providers is to offer flexibility to their customers.

Under some circumstances, the value proposition offered by pure players may fall short of complying with the RobMoSys goal of very long-term availability. In particular, pure service player differentiate from each other by building a core domain of expertise on a number of given technologies. If they are not able to differentiate sufficiently from competitors, they may stop supporting such technology and try to relocate their experts on other, non-RobMoSys technologies. This scenario is not unlikely in the current RobMoSys context, because the number technologies included in the RobMoSys perimeter is limited, whereas the number of pure service players is significant. If this happens, the RobMoSys ecosystem will include only a small number of pure service players, and the end-user will be confronted to a de-facto monopoly. In this case, this would be a high risk factor for end-users.

Customer Relationship

The customer relationship is a domain where RobMoSys can be of great help for service players. The fact that they obtain a label will facilitate relationship with the purchasing department of large corporations, for which the label will be a clearly-identified property of the service. The RobMoSys community could also provide, as part of its services for pure service players, a number of template contracts, adapted to commercial organizations, public entities, and so on, that could be used to standardize and facilitate the purchasing process.

Key Activities

As part of RobMoSys, pure service providers will have an important role as technology providers. This is mostly due to the small number of real Robotics Modelling editors in the domain of interest of the project. However, pure service are contract-oriented, and when contracts stops, they can lose the expertise acquired throughout its duration. In RobMoSys, long-term support requires to preserve expertise as much as possible. It is therefore critical than holding the RobMoSys label implies putting in place expertise-preservation activities, like participating to the development of the supported technologies out of any customer requirement. Of course, doing so has a cost.

Another important activity that pure service providers would have to carry out, due to the lack of Robotics Modelling editors in the RobMoSys domain, is the activity of industrialization of the software. The RobMoSys community will be used to manage the technology around RobMoSys tools. However, it does not involve managing products that the end-users can use. Bridging the gap between technology and product involves significant efforts: it implies testing and validating the technology on the platforms used by the customers, ensuring that documentation and certification material is available, making sure that all relevant technologies are properly integrated and packaged together. All these activities are critical, and need to be done. Pure service providers usually do it as part of a service contract, for one customer, and do not share the cost thereof on multiple customers. This is an obstacle to the RobMoSys goal of lowering the cost of software.

Revenue Stream

If expertise-preservation activities are put in place, they need to be funded. Given the pure service business model, this can be done only as part of a service contract. This would mean that RobMoSys-labelled services include by default a maintenance period, that would involve not only maintaining the development made for the customer, but also the technology as a whole. The contract templates could reflect that. Of course, it means that end-users would have to support the corresponding costs, but this cost reflects the cost of the corresponding service, which cannot

be provided for free. This would also inflect the business model of pure service providers and taking into account the key activities described above, would drive them towards a FLOSS leveraged services business model.

Key Partnership

Pure service players intend to specialize in a specific area of technology. This implies that, in this area, they will partner with the RobMoSys community, obtain the RobMoSys label and carry out the activities required to preserve this partnership. It also implies that, in the area where they do not specialize, they will partner with other pure service players with complementary skills, so that they are able to make comprehensive offers. Finally, they can also partner with leveraged services providers, acting as software editors, which would perform the industrialization activities described above and share their costs on multiple customers; they can also partner with infrastructure providers to make a comprehensive offer in term of development environment.

4.3.2 Leveraged Services

A leveraged service is an expertise-based service that facilitates the effective application of a free software product to specific business needs. This product/service combination is typically sold as a yearly subscription comprising the free software toolset with upgrades, high-quality toolset support and online consulting from toolset specialists. GNAT Pro, the free software development environment for the Ada programming language used in mission- and safety-critical systems is an example¹⁴.

The GNAT Pro offering illustrates the value proposition of this model. GNAT Pro customers receive a high-quality free software product accompanied by service directly from Ada experts and the GNAT Pro developers. The GNAT Pro experts serve as partners to the customer's development team. They work closely with the customer team to help them make optimal use of GNAT Pro and to assist them with all aspects of their Ada software development. As part of that dialogue, AdaCore engineers regularly address issues such as code optimization, programming language semantics, multi-language systems, and code organization. In addition to the benefits stemming from a high-quality product, this level of service leads to higher productivity and reduced risk in projects using GNAT Pro.

This business model leverages the subtle combination of two scarcities: scarcity of the free software expertise and scarcity of the free software quality assurance (QA) infrastructure. In the case of GNAT Pro, the GNAT Pro development team has built a sophisticated QA infrastructure comprising over 30,000 real life test cases (a total of over 6 million lines of Ada source code) and automated capabilities for execution and evaluation of the tests, supporting nightly integration, regression testing and analysis on multiple platforms.

This business model works for free software used in fairly to highly technical software projects requiring sophisticated tools, libraries, and other software components, along with a high degree of expertise, and of confidence in the reliability of the products. The software deployed in a dependable system is often of this sort. As a result, this business model is well-suited for industries producing highly-dependable software. What's more, this business model is in the interest of the companies ultimately responsible for the software used in dependable systems because it aligns their interests with those of their tool vendors. A company developing software for a dependable system is interested in receiving a high-quality product with high quality service in order to develop its dependable application. The only way a free software company can secure recurring revenue with this business model is by ensuring that the customer renews his subscription. This will happen only if the customer is happy with the technology and the service he receives: unlike for closed-software, the free software product alone provides no vendor lock-in.

The Leveraged Services business model can be summarized as displayed below in Osterwalder's

canvas:

Key partners Free software communities that design the software used in the product (e.g. GCC or Eclipse)	Key activities Integration of free software and redistribution back to community of the most important changes/fixes/enhancements. Productization activities to change the raw technology into industrial products.	Value proposition Supported products	Customer relationship Tight	Customer segments Large corporations
	Key resources Software developers for technologies used		Channels Internet support	
Cost structure Mostly in proportion of the number/complexity of the products.			Revenue streams Recurring subscription contracts.	

Customer Segment

This business-model involves providing high-end services. But such services are not affordable for consumers and even small companies, because providing them requires is a manpower-intensive activity performed by highly-qualified persons. This limits this business model to medium-to-large companies. In the context of RobMoSys, most of RobMoSys community members would be in this category.

Value proposition

It is a requirement to tightly link together complex software and high-value services. If you miss one, you don't belong to this business model. For instance, if the software is not complex enough, you are probably more like in pure service business model. If, on the other hand, the software is complex but you are not able to provide high-value services along with it, you may fall into the pure free software business model.

Revenue Streams

In this business model, revenue streams are subscription-based, with customers paying upfront for the duration of their subscription. This has a number of advantages, the most obvious being that this model is structurally sound cash-flow-wise, because money flows in at one time at the beginning of the subscription, while expenses are incurred throughout the provision of services, during the whole duration of the subscription. The critical point is to ensure renewal of the subscription, but some key activities are designed to achieve this requirement.

Key Activities

FLOSS puts far less legal restrictions for the use of software than proprietary software. In particular, FLOSS vendors cannot use the licence contract to prohibit users from using their product after their support agreement is terminated. FLOSS licenses also permit modification of the software, resulting in the impossibility for FLOSS vendors to use technical means to prevent users from using their product after their support agreement is terminated. To summarize, customers purchase proprietary software because they cannot do otherwise; they purchase FLOSS because they truly want to. The consequence of this is that FLOSS vendors must find incentives leading customers to renew their support contract. Things that can be done to achieve this goal include:

- (1) Having a very proactive R&D policy, so that new features get included in the FLOSS tools on a regular basis, or so that new tools are added regularly. If the technology evolves regularly, customers won't accept easily to get stuck with a given version.
- (2) Having a very good support service. Customers pay for it. They need to see that they get value for the money. For this reason, it is best to avoid anything that can limit the service provided, because customers could end up believing that they can get rid of it. For instance, it is probably not good policy to limit the number of reports customers can make, not mentioning that customer reports are a very good way to complement internal verification and validation with cases representative of real industrial use.

A number of specific activities need to be specifically performed by the RobMoSys members that implement this business model:

- (1) They need to obtain the RObMoSys label for the technologies included in the product they are selling.
- (2) They need to develop the product conforming to the RobMoSys requirements, including:
 - a) Conforming to RobMoSys legal policy when committing;
 - b) Committing changes on the RobMoSys repositories;
 - c) Planning for RobMoSys-required activities in the area of testing, documentation and validation;
 - d) Possibly, committing changes to other community repositories.

More generally leveraged services involve carrying out the industrialization activities needed to bridge the gap between the technology and the product, because customers expect the latter as part of the offer. Leveraged service providers know how to do that for multiple customers, therefore turning FLOSS technology into COTS. This coincides with RobMoSys' goal of lowering the costs, because the cost of this activity is spread evenly on all customers.

Key Partnerships

Obviously, the most important partnership in this context is the RobMoSys partnership itself, as it allows performing a specific service, expected by customers who are at the heart of this business model. For instance, the RobMoSys will provide an infrastructure which otherwise would need to be part of the key resources. RobMoSys partners are also key when providing services that are typically not performed or hardly performed by players implementing this business model. For instance, a pure service player can partner to provide deployment services for the tools marketed by a leveraged services player, or training services.

Customer Relationship

In order to have the high-level support service required by this business model, you need to have a tight relationship with your customers. This relationship starts at the beginning when customers are only prospects. You need a sales force that is able to find the needs of the customers, and given the high level of services that are provided, such sales force need to have a very strong technical background to elicit the customers' needs. Once the purchasing decision is made, you need to ensure that the customers are aware of the level of services they can get to maximize their understanding of the value offered, and therefore maximize the renewal rate. Finally, you need to gather information regarding the reasons for which customers do not renew their support contract, and keep a good relationship with them to be able to detect and win future opportunities.

Cost Structure

The Cost structure is almost completely the opposite of the revenue structure. Costs are incurred throughout the duration of the subscriptions. Actual costs may be different for each customer, as

the service actually provided depends on the customer himself. For instance, some customer may almost not use it, while some others may open a large number of support requests. In any event, for renewal purposes, it is important that customers are not limited in the number of requests they can make, and are incited to use the service as much as they can, and realize the value thereof. In addition, the bigger the number of support request, the more likely it is that some of them have to deal with the same issues, resulting in cost-savings.

4.3.3 OSS Co-Ops

The Eclipse Foundation and the Apache Software Foundation have an interesting and unconventional business model.

Eclipse is an open, extensible IDE (integrated development environment) built on a mechanism for developing, integrating, and running modules called plug-ins. Put it another way, Eclipse is a platform providing a common IDE infrastructure for tool providers to plugin their tools. The Apache Software Foundation provides support for the Apache community of open-source software projects such as the Apache HTTP server. These organizations leverage on the need for a single, unifying point in the development process of complex applications. To some extent, they demonstrate that the development paradigm of free software is not necessarily the bazaar: cathedrals can be built with hundreds of individual contributions.

Both the Eclipse Foundation and the Apache Software Foundation are non-for-profit organizations. They are characterized by collaborative, consensus-based organizations and development processes. They form tightly-coupled communities of developers and users governed by an open and pragmatic software license. We call these types of organizations OSS cooperatives or OSS co-ops.

A general-purpose co-op is an association formed and operated for the benefit of those using it. Its business purpose lies more in cost reduction than in revenue generation. The roots of the co-op concept lie in the work of Peter Kropotkin in 1902 [Kropotkin 1902]. In his book Kropotkin describes how in Siberia animals, instead of competing for resources, have to work together to stay alive. Throughout his book, Kropotkin stresses that cooperation is the main factor in evolution, rather than the competing forces posited by Darwin and his adherents.

It is precisely this drive for cooperation to solve a mutual problem, too hard or too costly to solve alone, that is the business case for OSS co-ops. The idea of industrial cooperation is not new. The Eurofighter consortium is a good example of cooperative effort across aerospace companies and nations. What is novel, and what the free software movement has shown, is that this idea can be applied to the development of software.

In this case, the revenue stream is made of a fixed fee that is computed according to various parameters. In the Eclipse foundation for instance, the fees depends on the revenue of the company for the so-called strategic developers. For the so-called strategic consumers, the fees depends not only on the revenue of the partner, but also can be reduced if some manpower is committed to the foundation. In return for their subscription fees, the partners get decision power on the future of projects hosted by the foundation, and are able to orient the strategy of the projects.

This model may be very well adapted to the RobMoSys, as this community would aim at providing key services to a population of users sharing the same technology, either industrial end-users or technology providers. It could require contributions in money or in kind (manpower), which amount would depend on the revenue of the partner, or the use of the services by such partner, or a combination of both.

The OSS Co-ops business model can be summarized as displayed below in Osterwalder's canvas:

Key partners End-users, providers	Key activities	Value proposition An infrastructure dedicated to the coordination of a key free software technology	Customer relationship Tight	Customer segments Corporation (paying) Individual hobbyists (free)
	Key resources Software developers Hardware and software infrastructure		Channels Internet support	
Cost structure Mostly in proportion of the number/complexity of the products.		Revenue streams Recurring subscription fees.		

We will not study FLOSS Co-ops anymore in the present document because the only relevant Co-op is the Eclipse foundation and the Polarsys platform thoroughly studied in RobMoSys deliverable D6.2.

4.3.4 The Proprietary Free Software Product

This business model is an oxymoron, or we could call it the have-your-cake-and-eat-it-too business model. In this model the software vendor assembles a set of free software and proprietary applications, and markets the bundle on the basis of perceived synergy among them. The vendor uses the conventional charge-per-unit model where the unit price can be higher because of the scarcity created by the proprietary components in the bundle. The value proposal is centred on the proprietary tool, and the free software component is only used as a way to increase the perceived value of this proprietary component.

Some GNU/Linux distributors have adopted, often temporarily, this business model. Another example of such a business model includes the Workbench development environment, which comprises an integrated development environment based on Eclipse along with proprietary development tools for the VxWorks proprietary operating system.

This model is so anchored in the conceptual milieu of proprietary software business that it offers marginal advantage to the customer over the conventional closed-software model. The only advantage of this business model for the customer is the access to the free software component within its bundle. However, the proprietary part offers often a real value and the free software component is of much lower value without it.

The Proprietary Free Software Product business model can be summarized as displayed below in Osterwalder's canvas:

Key partners Free software communities that design the accompanying free software	Key activities Design and implementation of proprietary software Integration of free software	Value proposition Key (proprietary) software	Customer relationship	Customer segments Large corporations
	Key resources Software developers for the proprietary sw		Channels	
Cost structure			Revenue streams Expensive professional prices	

So far no Polarsys member implements this business model.

Value-Proposition

The core of this business model is two-fold. First, there need to be a proprietary technology on which competition is only proprietary. In the WRS example, this is the VxWorks RTOS, which competition included, until recently, only proprietary RTOS like Green Hills Software Integrity. Second, there need to be a FLOSS technology, which can be used in combination with such proprietary technology and which is (i) a standard in the considered area, and (ii) is quickly evolving. In this case, unless you are a giant (and even in this case), spending time on developing an alternative is a waste of time: your resources will be diverted from the heart of your activity, and won't be sufficient to compete with a FLOSS alternative fuelled by a full and active community. Companies implementing this business model need to channel through the same canal both the proprietary core software and the FLOSS leveraged software. This is important because most of their added value is to provide a tightly integrated solution to their customers, and this can be done only if the two components are channelled in the same canal. This is also critical for the perception of the value of their products by the customers.

Key Partnerships

The most important partnership companies have in this business model is with the FLOSS community developing their tool. It is of utmost importance for such companies to be active members of this community and avoid "free-riding" on the technology. In order to gain legitimacy within the community, the company has to commit significant resources to this. This can be achieved by funding the community, or funding other players, but the best way to build up the partnership is to contribute efforts to the community development. As most of the communities are merit-based, the voice of the company will be heard, and it will be in position to influence the development of the technology in a direction compatible with its goals. If, on the other hand, it does not contribute effort to the technology, only trying to extract free support from the community, then it will have no legitimacy and will remain unable to shape the development of the technology.

It is important to notice that only the FLOSS leveraged product can be part of the Polarsys technologies, as all Polarsys-managed technical items are licensed under a FLOSS license. Therefore, the partnership with Polarsys is a critical one, and it is expected that companies implementing this business model do dedicated significant resources to carrying out Polarsys-specific developments. In addition, very long-term support of FLOSS applications may require other partnerships to be put in place (for instance, with another player that would accept to take over

exploitation of the technology in case the company did not want/be able to support it anymore).

Key Resources

In order to gain this legitimacy, it is therefore critical to gather a pool of developers specialized in this technology. This works as a virtuous cycle: when you start contributing, FLOSS developers will be attracted by your company and more likely to apply for a job; the company will become more attractive for developers and more likely to hire them. However, one should not forget, at the same time, that FLOSS is used as a way to promote some proprietary technology, and the savings obtained when adopting the FLOSS technology should be reinvested immediately in two activities: first the development of the proprietary software core, so that it remains cutting-edge; and the industrialization of the FLOSS technology, so that the two technologies are properly integrated in a well-documented, well-tested product.

Key Activities

Following the above, key activities include developing the FLOSS technology in tight relation with the community, in particular following the community procedures; performing industrialization activities on the proprietary core software and FLOSS leveraged software (like testing, integration, validation, qualification, certification, documentation, etc); following the evolutions of the community and be part of the community life to monitor the possible evolution of the technology.

A number of specific activities need to be performed by the RobMoSys ecosystem members that implement this business model:

- (1) They need to obtain the RobMoSys label for the technologies included in the product they are selling.
- (2) They need to develop the product conforming to the RobMoSys requirements, which entails:
 - a. Conforming to RobMoSys legal policy when committing;
 - b. Committing the changes on the RobMoSys repositories;
 - c. Planning for RobMoSys-required activities in the area of testing, documentation and validation;
 - d. Possibly, committing changes to other community repositories.

However, the FLOSS leveraged product they are providing is not the only element for which long term support needs to be provided. The proprietary core software product is by any mean as critical to their customers. It means that in order to take the full benefit of their RobMoSys membership, members implementing this business model will need to define a long term support methodology for the proprietary part of their offer.

The ways to do so remains open, but may be more complicated than for FLOSS components, as IPR issues are not solved, and expertise cannot be shared.

Customer Relationships

Software industry is the place of tough competition, where differentiation from the competition is direly needed. This can be achieved by mean of a differentiated product (including specific technology or high quality services), but this can also be done by mean of a differentiated relationship with the customer. This relationship starts when the customer is only a prospect. In the area of RobMoSys, where customers develop complex technology over long periods of time, where technical decision-making is slow, but reliable when secured, there is an opportunity to build a high-quality sales force. In this case, it is important that sales persons have a strong technical background, so that they can help customers solving their problems, rather than just advertising the product, and can be perceived as reliable persons that can be referred to by people who are

fond of the technology. In this case, the traditional boundary between sales force and technical team is not relevant, and there should be a continuum between the two.

Revenue Streams

Most of the companies selling proprietary software have a pay-up-front (PUF) model. If they sell FLOSS software as part of their offer, this component is provided for free, and is paid for by the licence fee paid for the proprietary core product. In addition, for those selling software embedded in the customer product (typically a middleware or RTOS) they may also require a royalty fee for each product sold. This system is not an incentive for long-term relationships. In particular, the PUF model does not encourage people to renew, because the provider is not incited to provide good maintenance (the fee for maintenance is too low) and the customer, having poor service during the maintenance phase does not want to renew it. As a result, the provider is incited to increase the licence fee, closing the loop of de-incitation. We believe that a good way to go out of this nasty loop is too move towards a business model close to the Leveraged services, where fees are even throughout the subscription, and where the offer is increased throughout the relationship.

4.3.5 Dual Licensing

In this model, the free software work is distributed at no cost under the terms of the GPL or any other non-permissive copyleft license, as well being sold under a different license to customers who do not want to be bound by the terms of the GPL. Of course, this model is available only to the copyright holders of the software, as only the copyright holder can change the license of the software. The value-proposition leverages on product scarcity.

It is viable only when the software being sold is included, in whole or in part, in another application that is in turn sold by a customer under terms different from the GPL. If a customer were to include in his own work software licensed under the GPL or any similar copy-left non-permissive license, then his application would be considered a derivative work of such software, thereby forbidding redistribution unless the whole application were licensed under the same license. MySQL, a database, and Cygwin, a UNIX-like environment for Windows, were licensed this way. This is an interesting illustration of the fact that the copyright holder can distribute the same version of a software work with different licensing conditions.

With respect to dependable systems, this business model rests on vendor lock-in. It has the advantage, relative to the completely proprietary model, of offering the sources of the product at no extra cost. It may also leverage a larger community of users and perhaps additional expertise independent of the vendor. Apart from these, it offers no significant advantage over a proprietary software model. Its appropriateness to industries developing dependable software depends more on the relevance of the product and associated services than on any enhancement of value due to the business model.

Also note there is a variant of this business model, that is not a truly free software business model, where a crippled free ("as in free speech") software version is made available for free ("as in free beer"), while the full-fledged version of the software is made available to paying customers under a proprietary license.

The Dual license business model can be summarized as displayed below in Osterwalder's canvas, keeping it mind that it addresses at least two customer segments, one for the free software product (1) and one for the proprietary-licensed product (2):

Key partners	Key activities	Value proposition - Free software -Proprietary software	Customer relationship - Loose - Tighter	Customer segments -Academia, hobbyists corporations in exploratory mode - Corporations which make product incorporating the software
	Key resources		Channels	
Cost structure			Revenue streams - Commodity prices or even free - Expensive professional prices	

In order to be in position to license the same software under different licenses, the vendor needs either to be the copyright holder on the software or to gather the consent of copyright holder(s) to do so. This consent is practically impossible to get a posteriori when contributions are coming from significant number of different people, which means that this business model is limited to vendors working in a non-collaborative fashion, or those which get the consent of authors ex ante. Also, this model works when customers need the more permissive terms to implement their own business model. In most cases, it means that they need to be able to incorporate the FLOSS provided by the vendor into their own (often proprietary) software. This restricts the interest of this business model to vendors of software embedded into the final application (typically middleware, libraries, building blocks, etc.) rather than tools. For this reason, it is not likely to be a widespread use into the RobMoSys context.

4.3.6 Infrastructure Provider

Companies such as Geeknet Inc. leverage scarcity of infrastructure to provide an OSS development website, SourceForge with a large repository of free software projects for whom the basic service is free of charge, while advanced services come at a fee. Geeknet delivers more than 2 million downloads a day and reaches 46 million unique visitors per year. On the strength of this network effect, some of its revenue is indirect from web advertising. Other online revenue comes from per-user annual subscriptions that give access to advanced services on the SourceForge.net web site, such as advanced search capabilities, priority technical support, project monitoring, etc. The annual cost per user is low (less than USD 50 in 2004).

Geeknet also used to sold SourceForge Enterprise Edition to manage and execute offshore and distributed team development. The SourceForge application itself does not appear to be free software. Geeknet leveraged the free service it provided to the free software communities to sell and promote its SourceForge product. However, later on, Geeknet decided to put SourceForge under a proprietary license, and sold it to Collabnet, which markets it now under the trademark TeamForge. However, forks of the technology occurred that still allow access to it under a free software license.

This business model is nevertheless interesting for developers of dependable system that have a large and possibly distributed development teams. Leveraging the experience with collaborative development in the free software community and using the same infrastructure that was created to support it can be an attractive solution, in particular in an industrial context where the design of embedded systems is more and more distributed, leveraging on subcontractors, some of whom may be located in other countries like India.

The Infrastructure Provider business model can be summarized as displayed below in

Osterwalder's canvas:

Key partners Free software communities that design the configuration management, the bug tracker, etc ...	Key activities Integration of free software Development of forge software	Value proposition Integrated infrastructure	Customer relationship - Very loose (free customers) - Tight (paying customers)	Customer segments - Hobbyists, academics (free customers) - Large corporations
	Key resources Software developers for the forge		Channels Internet access	
Cost structure Hardware is an important cost factor			Revenue streams - Advertisement fees for free customers - Expensive recurring professional prices (subscription) for paying customers	

Value-Proposition

In the frame of RobMoSys, the kind of low-grade free services offered by infrastructure providers to the general public are out of the scope. It means that the value proposition has to include high-end services (like build farms targeting popular embedded processors like PowerPC or Arm, test beds emulating the final environment, etc), and warranties like 24/7 guaranteed service, confidentiality assurance, etc.

The offer must ensure that the adequate levels of confidentiality and security are met. It also means that, beyond any contractual warranty, which may not prove effective to protect customers against the possible actions of foreign intelligence services, servers are located in trusted countries. Also, infrastructure providers may have to deal with personal data and should comply with the corresponding obligations.

Customer Relationship

Given the highly-customized kind of services that are provided, the level of relationship with customers must be much higher than it is in regular infrastructure providers. In particular, customers must have direct access to the support hotline to ensure that the warranties they pay for are effective.

Key Activities

Key activities include maintenance of the infrastructure. Usually, infrastructure providers know very well how to maintain large farms of servers and ensure constant availability of them. However, this is usually achieved by a high level of standardization of the hardware and software used on such servers. In this case, the high level of service customization will require finding innovative solutions to keep the same level of QoS. This will probably imply using systematically virtualization techniques. Key activities also involve the monitoring of the tools and hardware technologies to determine when be discontinued, and find solution to address this situation. With respect to FLOSS leveraged hardware vendors, which do the same kind of monitoring, infrastructure providers will do this at a higher level. For instance, a hardware vendor will monitor if the processor is discontinued; the infrastructure provider will monitor if workstations of a certain type are discontinued.

Key Partnerships

The infrastructure provider will need to partner with the RobMoSys community to ensure it is aware of the changes in the RobMoSys technology. However, it may need other partnerships, most probably with a pure service player, so that it has the resources to customize the RobMoSys technology for use by end-users. This may be the occasion of moving pure service players from one-shot, short term service contracts to long-term or recurring contracts, as the infrastructure provider is more likely to need them continuously if it has enough customers. Infrastructure providers may also partner with FLOSS leveraged hardware vendor(s), if any appears, to build their infrastructure and ensure very long term availability thereof. In this case, their offer will need to reflect the conditions made by these vendors in terms of duration.

Revenue Streams

A key element in this business model is to find how to price the services. This involves defining a pricing model, which may depend on many factors: number of users at the client's premise, level of availability, level of long term support (from "none" up to "we warrant that you will have x86 servers available during the next 30 years"), number of servers needed, etc. Given the huge investments to put such an infrastructure in place, it is required to price it appropriately, but not at a prohibitive level, otherwise people will implement the service on their own.

4.3.7 Free Software Leveraged Hardware

With the advent of 64-bit chips such as the Itanium from Intel, a number of hardware vendors have announced 64-bit workstations running (sometimes customized versions of) Linux. An interesting example is the SGI Altix platform. In this business model hardware vendors leverage the Linux "network effect" to offer high-performance servers at premium prices. In this case, the value proposition is based on the provision of the expertise required to optimize the use of the operating system on a given hardware; communication channels must be adapted to the fact the hardware is provided, which involves much higher logistics.

Bundling a free software operating system (OS) and related software components, along with services covering the complete platform, is an interesting offer to developers of dependable software: the entire system on which their application rests becomes open for inspection. The QA and other documents typically needed when developing dependable software are not available, but for certain types of dependable systems having access to the OS source code allows a dependable system vendor to create an in-house team to qualify and tailor the OS according to the dependable system requirements. The additional support provided by the hardware vendor for the platform (and typically bundled into its premium pricing) is a differentiating factor for customers developing dependable systems, compared to commodity hardware with little or no vendor support.

The Free Software Leveraged Hardware business model can be summarized as displayed below in Osterwalder's canvas:

Key partners Free software communities that design the OS	Key activities Integration of free software Contribution of hardware-specific code to the OS (drivers, optimizations ...)	Value proposition Transparent hardware+software bundle, with services	Customer relationship Tight (in particular when dedicated services)	Customer segments Large corporations
	Key resources Software developers for the proprietary sw		Channels Internet distribution for software plus conventional channels for hardware.	
Cost structure			Revenue streams Expensive professional prices	

Value-Proposition

The vendor need to have some hardware it wants to promote. In the case of RobMoSys, which focuses on development tools, hardware could only involve development workstations. Large vendors, like IBM, Oracle (formerly Sun) or HP, usually provide these workstations. RobMoSys could be an occasion for other players to target the aerospace market by providing software-hardware combos including the RobMoSys tools on one hand, and ordinary x86- and Linux-based hardware on the other hand. In addition, there could be a RobMoSys-specific offer to provide such hardware during extended periods of time, so that very long term support is possible.

Revenue Streams

Revenue streams would be related to the selling of hardware, and be one shot. However, if the vendor provided RobMoSys-specific services (like provision of hardware over very long periods of time), subscription-like services could be considered.

Key Partnership

An important factor in such an offer is the software part of the offer, which is typically not at the heart of the vendor's activity. In order to maintain this software component, partnerships would be needed, with other RobMoSys players for the tool part of it, and with Linux vendors like Red Hat or SuSE for the operating system part of it. Also, if a very-long term support of hardware is considered, specific partnerships with the hardware manufacturer of key elements (like processors for instance) would be required so that they are able to provide the corresponding hardware over extended periods of time.

Key Activities

A key activity here will be the active monitoring of the hardware components that are likely to be discontinued by their manufacturer, and finding solutions to either replace them in a way that does not prevent supporting the underlying software, or stock-piling them for their expected duration of use. This implies probably that vendors need to have some visibility on the duration of subscriptions, hence long subscriptions (typically 5 years) which renewal would have to be advertised well in advance.

4.3.8 Pure Free Software Product

In this model the free software vendor limits itself to selling a standalone, self-contained set of free software programs with perhaps some minimal installation help.

As for any software product, there is no restriction on the price that can be charged for software licensed under a free software license. Recipients of free software can buy one copy of the software and install and run it on as many machines as they like. This means that the conventional sell-per-unit business model does not work very well for free software since free software licenses favor an economy of software abundance. Furthermore, recipients of free software programs have themselves the freedom to redistribute copies of the software product and to charge for it. This limits the amount that the initial vendor can charge to pretty much commodity prices, unless the free software product is both one-of-a-kind and sold in low volume to address specialized business needs.

Thus, value proposal of this model leverages solely on scarcity of infrastructure (assembling of proper free software packages and redistribution). Most commercial GNU/Linux distributions until recently were based on this pure free software product commodity price model, where communication channel with customers are as economical as possible (typically download from a dedicated web server). Because the infrastructure required for this kind of operation is easily available, this type of business model has been abandoned by a number of GNU/Linux distributors.

For developers of dependable systems, this business model is not particularly attractive or relevant. Development of dependable software is an expertise-intensive task usually employing sophisticated technology. As such, downloading binaries at commodity prices with little or no associated service is not particularly interesting for two reasons. Either the free software product is a sophisticated toolset, in which case it is critical that high-quality service accompany the product, regardless of its free software status. This means only that the real value proposition of the product must include the service, and that the underlying business model shouldn't be the pure software product. Alternatively, software is a component, such as a library or operating system to be embedded in the final dependable system, in which case getting a CD containing sources is nice, but not nearly enough to meet the customer's need for evidence. In this second case, the real value proposition should include the evidences as well. In real-life, dependable systems often combine these two requirements.

Overall, the pure free software business model can be summarized as displayed below in Osterwalder's canvas:

Key partners Free software communities which software is integrated	Key activities Integration+construction of software	Value proposition Commodity software (OS, office software ...)	Customer relationship loose	Customer segments General public, corporations that don't want to pay much on software
	Key resources Hardware infrastructure		Channels Internet distribution	
Cost structure			Revenue streams Low commodity prices, need to be compensated by high volumes.	

This business model is not attractive for providers of high-value technologies, because such technologies come along with high-value services that have no room in this business model. In addition, the major players are abandoning this business model because it does not allow garnering significant revenues. So it won't be studied anymore.

4.4 Business Models per Partner

4.4.1 Summary of RobMoSys partners view

The RobMoSys partners are in favour of open innovation models and committed to sharing results in open-source as enabled by the Eclipse Public License (EPL). This holds especially for the metamodels, models, software components and prototypical tooling of the RobMoSys platform and interface transformations with other related technologies, with the goal of reinforcing the RobMoSys open community for its further development and maintenance. RobMoSys partners are familiar with open-source ecosystems, especially the Eclipse, ROS and Gazebo ecosystems and the associated open-source licenses and open document licenses (e.g. Creative Commons).

For the core-development, RobMoSys partners are in favour of any permissive licensing model that allows to

- an active development contribution from the community;
- a commercial use of the software product by third-parties;
- clear acknowledgement of the contributions and authorship.

Concerning EPL V2.0, it has been designed to promote collaboration on the open source platform, while at the same time enabling various business models, including the creation of proprietary products for adopters.

The question which license would be most user friendly from an industrial user perspective would need a more thorough research. We plan to include this question in the workshops to be organized with external experts and other RobMoSys stakeholders.

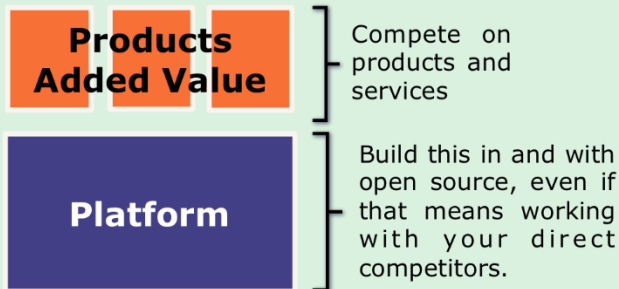
4.4.2 Questionnaire feedback

In this section, we present a summary of the planned business models per RobMoSys partner.

Partner	HSU
Type of Organization	HSU is at first "Research & Education", in particular within RobMoSys. However, HSU is also a "Tool and Service Provider" and a "Software Developer" as we provide free prototypical Open Source Tooling for Model-Driven Software Engineering (SmartSoft, SmartMDSD) as well as free prototypical Open Source software components for service robotics applications.
Most appropriate licensing model	<ul style="list-style-type: none"> • structures, meta-models, ...: <ul style="list-style-type: none"> ◦ free and open license which requires no fees or memberships to read them and which allow everybody to build whatever kind of tools etc. conformant to these structures • prototypical tooling as we do it in RobMoSys: <ul style="list-style-type: none"> ◦ free and open source license for the tooling such that this license is compatible with typical tool development environments, in case of HSU this is BSD which is compatible with the Eclipse license ◦ this is not in conflict with commercial tools with their own license which does not need to be free and open source • software components built with the tooling:

	<ul style="list-style-type: none"> ○ whatever license you want, ranging from software components that are free open source (even GPL) to proprietary licenses ○ you should be able to select any suitable license for what the tooling generates in order to be able to build whatever licensed components with the tooling
--	---

Partner	COMAU
Type of Organization	Industrial (Robotics).
Most appropriate licensing model	BSD

Partner	ECL
Type of Organization	Open Source Foundation.
Most appropriate licensing model	<p>We suggest promoting software licensed under the Eclipse Public License V2.0, eventually with dual licensing under the BSD license in order to spread adoption of the software in different open source contexts.</p> <p>Concerning EPL V2.0, it has been designed to promote collaboration on the open source platform, while at the same time enabling various business models, including the creation of proprietary products for adopters. See the following figure.</p> 

Partner	KUL
Type of Organization	Research & Education.
Most appropriate licensing model	<p>Our contribution is rather simple: all our efforts are focused on making an ICT platform that is fully independent of any business context and has no commercial exploitation purposes in itself. The business motivation to spend this effort is then to allow others to build business models on top of that neutral and open platform. The situation is 100% comparable to the efforts that many stakeholders (academic, industrial, governments) have put, and are still putting, in the creation, expansion and improvement of "the web", that is, by developing the standards, reference implementations and tools around HTML, SVG, Javascript, CSS,</p>

WebRTC, WebSockets, HTTPs, AJAX, JSON-LD, GraphQL, RDF, etc. In that context, the HTML is the "component container" in which all of the others can be deployed, and HTTPS is the "communication middleware" via which all the components can interact with each other. RobMoSys is doing the same thing but at a higher level, that of robotics functionalities and architectures (composition and communication).

Concretely, for the core-development, any permissive licensing model that allows to

- an active development contribution from the community;
- a commercial use of the software product by third-parties;
- clear acknowledgement of the contributions and authorship.

Some examples of open-source licenses that allow such a model are: MIT, BSD, LGPL.

For extra-tools or functionalities, in terms of plug-ins, specific application-dependent configurations or by other means, a dual-licensing model is possible, depending on the software product user needs: an open (or free) license for no-profit usage, and a commercial license model for profit usage.

Partner	CEA
Type of Organization	<p>Research & Education.</p> <p>CEA is a Research and Technology Organization. Two laboratories of CEA are involved in RobMoSys project:</p> <ul style="list-style-type: none"> • The Interactive Robotics Lab (LRI) is specialized originally in remote handling for operations in hazardous environments (nuclear, underwater applications), the LRI has now a large part of its activities devoted to manufacturing. The laboratory has an historical background in robotics for Healthcare applications including assistive, surgical robotics and rehabilitation robotics. One part of the laboratory's activities dedicates to field robotics essentially for agricultural applications. Its main research foci carry on human robot collaboration (co-working). The technological researches cover mechatronics and the conception of actuators for the design of innovative robotics systems, robot control, supervision and user assistances (force, haptic, vision, graphic, immersive feedback) aiming at an efficient human robots collaboration in all domains of applications. In RobMoSys, LRI plays the role of user of the RobMoSys approach. • The Model Driven Engineering Lab (LISE) federates research on software and systems engineering, with a special focus on the design and validation of complex, critical system and software over the project life cycle. Cyber-physical System design activities are anchored to a general model-driven engineering platform: Papyrus an Eclipse-based open source UML-compliant software design suite. In RobMoSys, LISE plays a role of method and technology provider of the RobMoSys approach.

Most appropriate licensing model	<p>In order to maximize benefits of the RobMoSys approach, it is preferable to avoid non composable licensing models (e.g. contaminating licensing like GPL for instance).</p> <p>The licensing model depends on the nature of components of the RobMoSys approach:</p> <ul style="list-style-type: none"> • RobMoSys meta-models: fully open and business friendly licensing model (e.g. Eclipse Public License, BSD, ...); • RobMoSys component models: no preference; • RobMoSys tooling: depending on the tooling feature, but there must be at least open and freely accessible model editors.

Partner	EUnited
Type of Organization	<p>EUnited does not really fit well in any of the proposed categories. As an industry association, our main task is to support our member companies, which are robot manufacturers (mainly industrial, but also a few service robotics companies) as well as component manufacturers.</p> <p>Our focus is pretty much on industrial robotics.</p>
Most appropriate licensing model	<p>We would not grant licenses ourselves – most likely our members would use open source software under licenses granted by one of the RobMoSys partners.</p> <p>Using open source licenses in commercial environments seems to be a complex topic. Apparently, there are some pitfalls for companies using open source software licensed under different licenses in complex products.</p> <p>The question which license would be most user friendly from an industrial user perspective would need a more thorough research. If this is what would be useful from RobMoSys consortium perspective, we could address this question.</p> <p>On the other hand, from the discussions led by the different partner' lawyers, it seems like there is no real discussion about licenses, as every entity has strict rules on which licenses they would grant and which not.</p> <p>It could be worthwhile to discuss this more in detail.</p>

Partner	PAL
Type of Organization	Industrial (Robotics).
Most appropriate licensing model	<p>PAL Robotics has contributed actively to open source: https://github.com/pal-robotics</p> <p>PAL Robotics software are built on top of open software: Linux (GPL), GNU C library (GPL), ROS (mostly BSD, LGPL and GPL).</p> <p>Thanks to the open source community the time to market and the costs</p>

	of robot production has been reduced dramatically. The main consequence is that also a SME can afford to enter the market and be competitive.
--	---

Partner	Siemens
Type of Organization	Industrial. Tool and Service Provider. Software developer. Siemens “Corporate Technology” creates new market opportunities through interdisciplinary forefront work on promising technological fields and transfers the achieved results in close collaboration with the operative business units into innovative products.
Most appropriate licensing model	The license model should be permissive (non-copyleft) to allow the inclusion of the software as part of programs distributed under other licenses, including proprietary licenses

Partner	TUM
Type of Organization	Research & Education.
Most appropriate licensing model	We would prefer a simple, well-known permissive license (e.g. MIT or Apache 2.0), which would facilitate adoption of the software developed within the project (and after its end) without limiting commercialization based on providing services around the RobMoSys supply-chain. However, from the political point of view it may be interesting to adopt the European Union Public License, which is compatible with many other licenses (including MIT and Apache 2.0) and adds the “European Flavour” to RobMoSys.

5 Conclusions

At this moment the RobMoSys ecosystem is currently under definition and its exact form is still debated. Nevertheless, the aim of the RobMoSys ecosystem is clear, and in this context we have looked at the future RobMoSys market and analysed its business opportunities. The goal of this deliverable is to provide a baseline for exploring the added value of the possible RobMoSys business cases.

First we have looked at the market opportunities and have identified a set of market trends, risks and opportunities. This provides a baseline to explore more details of the RobMoSys market in the next iterations of this deliverable. We will conduct market surveys to complete the current market analysis.

We have then identified some business models based on open source. In particular, business models associated to the RobMoSys "products" are defined using Business Canvas. For all of these business cases we have demonstrated what the aggregated business process looks like, and how the stakeholders involved benefit from the future RobMoSys ecosystem and community.

Not a single business model is able to address all the needs that may appear inside the RobMoSys ecosystem. Partnering is required to cover all the needs, which is a change with respect to the traditional competition model.

The potential RobMoSys business cases provide enough support to claim that the future RobMoSys ecosystem has, from a business perspective, a clear and feasible goal.

6 References

- [Osterwalder 2010] Alexander Osterwalder, Yves Pigneur, "Business Model Generation", John Wiley and sons, 2010.
- [Grefen 2010] P.Grefen, "Mastering E-Business", Taylor & Francis, 2010.
- [Wipo Web] <http://www.wipo.int>
- [OpenSource Web] <http://www.opensource.org/>
- [FSF Web] <http://www.fsf.org>
- [Osami 2011] OSAmI Consortium, "Business problems and process approaches", 2011.
- [Roadmap 2015] Robotics 2020, Multi-Annual Roadmap For Robotics in Europe, Call 2 ICT24 (2015) Horizon 2020, 2015, https://www.eu-robotics.net/cms/upload/downloads/ppp-documents/Multi-Annual_Roadmap2020_ICT-24_Rev_B_full.pdf
- [Technavio 2016] Technavio, "Global Robotic Software Platforms Market 2016-2020", 2016.
- [Girard 2016] A. Girard and C. Rommel. The Global Market for Software and System Modeling Tools, 2016. IoT & Embedded Technology. <http://www.vdcresearch.com/Landing/iot1/A-15-Software-Systems-Modeling-Tools.aspx?bms.tk=BzAEqwsIp20Hm21Vr30SI33Ss26Bk17Ms20BvfrFtg>
- [Eclipse 2016] Eclipse Foundation. Developing an open source MBE tool suite based on the Papyrus platform, 2016. <http://www.polarsys.org/ic/papyrus>
- [OPEES 2013] OPEES Project, "OPEES Deliverable D1.2.1 - Open-Source Business Models for Polarsys", 2013.
- [SPARC 2013] The SPARC Partnership for Robotics in Europe, "Strategic Research Agenda for Robotics in Europe 2014-2020", 2013.
- [IFR-WorldRobotics 2017] International Federation of Robotics, IFR Forecast 2017.
- [Murphy 2017] Andrew Murphy, "ROBOTICS SOFTWARE AND SERVICES IS WHERE THE TRUE VALUE LIES", 2017.