RobMoSys

H2020—ICT—732410

RobMoSys

Composable Models and Software for Robotics Systems

Deliverable D7.3

Sustainability Plan

Huáscar Espinoza (CEA), Gaël Blondelle (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: D7.3

Document title: Sustainability Plan

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 (EFE), Susanne Bieller (EUR), Anna Principato (TUM)

Reviewer: Herman Bruyninckx (KUL)

Executive Summary

This document represents the first annual deliverable of Sustainability Plan (D7.2) for RobMoSys, as part of work package "Exploitation" (WP7). This deliverable aims at planning the RobMoSys related activities beyond the end-of-the project.

The main topics of this document are:

1. Background on sustainability concepts, as well as on other community management aspects.
2. Preliminary strategy for sustainability focused on the RobMoSys open source community, including a strategy for a sustainable business, quality management, maturity and long-term longevity of RobMoSys tools and assets.
3. Proposal for the governance of the RobMoSys community, as a first approach to be validated by key external actors, which includes governance principles and rules, decision making process, committees, and community tools.

This first version of the Sustainability Plan is focused on establishing the basics and on specifying a framework for community management and provision of activities for ensuring a strong sustainability of the RobMoSys results. This scope will be extended in future annual iterations by adding the feedback from Tier-1 experts coming from the community management 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:

- From the early stages of RobMoSys, a strong candidate for the open-source community, and in particular for hosting the tool assets, has been the Eclipse ecosystem. We must assess this option and any related mechanism around this option from the second year by taking into account that the innovation required to advance RobMoSys ecosystem needs to be driven by the key industrial companies.
- RobMoSys partners expect that the project solutions will evolve in pace with the more challenging requirements of modern robotics systems and will provide more flexible extensibility and customization that makes it easier to adopt the tools to the methods and processes of robotics engineering teams.
- It is also important to note that it is essential to sustain RobMoSys activity in communities that are outside of the RobMoSys community. In order to make the RobMoSys activity sustainable after the end of the project it is also necessary to build a dynamic, growing community around it.

Content

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.

The creation of a **sustainable ecosystem** is a major objective of the project, but obviously, this is not something that can be done in a guaranteed or strictly controlled way. However, the more clever, more open and more reactive the strategy is, the better this goal is to reach. Anyway, the project partners though will very actively stimulate active involvement of, and controversial discussions with, a selected variety of different stakeholder groups outside of the consortium of the project representing both technology-pull (application domains) as well as technology push (technical capabilities).

Sustainability is directly related to the business opportunities (see Deliverable D7.2 for full details) to produce RobMoSys artefacts (models, software components and tools). The effects of the commoditization strategy pursued in RobMoSys must be carefully analysed during the project-lifetime. Typically, commoditized software is a software nobody pays for (or is not aware to pay for, as in the case of operating systems). The sustainability of commoditized software directly relies then on open-source communities, and in more structured way through industrial foundations/group of interests. Sustainability through the creation of a foundation or group of interest in existing foundations is being investigated during the project life-time with the final objective of keep sustaining the community.

This document addresses sustainability issues of the RobMoSys ecosystem, particularly focused on an open source and open assets strategy. The proposed sustainability strategy and framework will be incrementally specified and continuously aligned to the exploitation and business strategies. This deliverable represents the first annual iteration of **Sustainability Plan (Deliverable D7.3)**, as part of work package **"Exploitation" (WP7)**. It aims at organising and planning the RobMoSys related activities beyond the end of the project.

1.1 Scope

This first version of the Sustainability Plan is focused on establishing the basics and on specifying a **framework for community management and provision of activities** for ensuring a strong sustainability of the RobMoSys results. This scope will be extended in future annual iterations by adding the feedback from Tier-1 experts coming from the community management 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.

1.2 Document Structure

The remaining of this document is organised as follows:

- Section 2 provides a background sustainability concepts, as well as on other community management aspects.

- Section 3 presents a preliminary strategy for sustainability focused on the RobMoSys open source community, including a strategy for a sustainable business, quality management, maturity and long-term longevity of RobMoSys tools and assets.
- Section 4 addresses a proposal for the governance of the RobMoSys community, as a first approach to be validated by key external actors, which includes governance principles and rules, decision making process, committees, and community tools.
- Section 5 summarises next steps and conclusions.

2 Background

2.1 Basics of Sustainability

Sustainability is the ability to develop and implement technologies/methodologies, which are self-sustaining without jeopardising the potential for future generation to meet their needs [Commission 1987]. An open source approach is good for sustainability of a project as it is an enabler to attract a larger community of developers and adopters, as well as a guarantee that if the existing developers of a project decide to leave, the code will still be available for the community to take over the project.

Governance relates to consistent management, cohesive policies, guidance, processes and decision-rights for a given area of responsibility. The management or governance of an open source community is directly related to the type of this community and what kind of product or result it offers to users. It is therefore a critical aspect to be taken into account when creating an open source community. How the community is managed and the type of licence hold is the key aspect when talking about open source communities.

An **Open Source community** is the keystone for the sustainability of a project. If the project development team is not able to attract and convince people that the code is worth spending time and resources on testing it, providing feedback, providing patches, and contributing in general, then a large part of the intrinsic value of Open Source is lost.

In other words, without **Maturity, Quality, Cost of Acquisition and Contro**l, the sustainability of the code is nearly impossible. And vice-versa, a community of adopters, testers, users, extenders of a technology is a great indicator demonstrating the Maturity, the Quality and the Control of the code.

Metrics: A good way to know if an Open Source project is sustainable and viable, is to check the activity of its community: number of committers, number of commits, regularity of the releases, and the quality and quantity of assets built around the project are a great indicator. In other words, the community is an excellent evaluation metric for a project.

There is a 'snowball' effect: a project attracts early adopters with the quality of code, the initial assets attached to the code like the Getting Started guide, documentation, scientific and technical papers, first releases, well defined code infrastructure (bug tracking system, continuous testing, continuous integration, etc.), and the interest of the adopters will do the rest.

2.2 Governance Models of a Community

How an open source project is managed defines the type of community and product that it will create. It is therefore a critical aspect to put some thought into the suitability of the governance model of robotics software that RobMoSys wish to adopt or develop.

The governance model can be taken in mind from different points of views. In general terms the governance can be classified in two different points of view, a centralized one and a de-centralized one. This classification is sufficient and there is no need for anything else when the object to be governed is a small piece of software where few people are involved. But when the project gets bigger the needs of a correct governance model becomes a necessity.

For the fulfilling of the RobMoSys necessity and following this "centralized" - "de-centralized" model, we can define two main approaches of classification, the **Cathedral/Bazaar** one and the **Meritocracy/Benevolent-Dictatorship** one. These two classifications can be considered at the same time for the same project, being the first one a general governance approach and the second one a definition of collaborating roles.

Table 1 provides a general classification of governance models.

Table 1. Community Governance Models

Classification	Model
Traditional	The Cathedral Model
	The Bazaar
	Meritocracy
	The Benevolent Dictatorship
Depending on the orientation of the community	Exploration oriented
	Utility oriented
	Service oriented
Depending on collaborative entities organization	Enterprise model
	Foundational model
	Consortium model

2.2.1 Traditional classification

The Cathedral model

The source code is available with each software release, but code developed between releases is restricted to an exclusive group of software developers. GNU Emacs and GCC are presented as examples.

The Bazaar model

The code is developed over the Internet in view of the public.

Meritocracy

A meritocratic governance model is, at its most extreme, the one that appears to give control away to community members in response to their contributions to the project. The Apache Software Foundation, perhaps the most famous example of a large scale meritocratic community, is very proud of the fact that they operate with an almost completely flat structure. However, even this model is designed so that those with control today can decide who gets control tomorrow. In this way project leaders ensure that only those sharing a common vision and, just as importantly, the willingness to work towards that shared vision, are given decision making authority. The "flatness" comes from the fact that once someone has decision making authority they have exactly the same authority as everyone else. Another aspect to the flatness of a meritocratic project comes from the fact that decision making responsibilities are usually reserved for those willing and able to understand and appropriately represent the views of the wider community.

Meritocratic projects often start life as a small number of decision makers, possibly even a single person, who provide a mechanism for the distribution of control from the dictator to a fully flat structure in recognition of contribution. Thus even a meritocratic model may look and behave like a benevolent dictator model in the early days.

The debian Linux distribution is the best example for this governance model.

Benevolent-Dictatorship

In a Benevolent-Dictatorship governance model, Project founders who maintain individual lead

throughout the entire life of the project are sometimes called 'benevolent dictators'. A benevolent dictator is responsible for providing the general direction of the project and making the final decisions when the community is in disagreement. As more and more members join the community, the benevolent dictator strives to ensure that these decisions are in the best interest of the project, rather than of any particular individual or institution. A good benevolent dictator needs to be able to balance any conflicting requirements of the community members.

The Linux Kernel development is the best example for this governance model.

2.2.2 Governance Model depending on the Orientation of the community

One of these classifications, that take in account the previous general subdivision, could be defined depending on the orientation of the collaboration, having three different governance models:

Exploration Oriented Governance Model

The objective of this kind of collaborative governance model has the main focus on the share of knowledge and innovation. It's driven by a Cathedral like control, were there is a central and main control. This governance model defines a community structure that defines a main Project Leader and the rest of the roles involved play a “reader” role. Examples of this governance models are, GNU, Perl and the Linux Kernel.

Utility Oriented Governance Model

The objective of this second kind of collaborative governance model has the main focus on the satisfaction of an individual need. It's driven by a Bazaar like decentralised control, where there isn't any core or main control point. This governance model defines a community structure that includes many roles from peripheral developers to passive users. Examples of this governance models are GNU Linux (parts that is not part of the Linux Kernel).

Service Oriented Governance Model

The objective of this third kind of collaborative governance model has the main focus on the provision of a stable service. It's driven by a Council-like centralised control. This governance model defines a community structure that includes a bunch of leaders instead of a unique Project Leader and there are clearly identified the rest of users that develop systems that provide a concrete service to the end users. Examples of this governance models are, Apache and PostgreSQL.

2.2.3 Governance Model depending on the collaborative entities organization.

Continuing with these more detailed governance model subdivisions a briefly used classification is this one:

Enterprise model

In this model the promoter (organization or partnership) decides to lead a project of Free Software where the Managing Committee will have the following characteristics:

The promoter will maintain the control on the decisions of design and management of the project naming or integrating the Managing Committee of the Project.

The promoter becomes leader from the beginning of the project, and this one is the unique one that can yield this position in case of considering opportune to leave the project.

The leader reserves the right to change this model in case he considers it necessary and with the condition of which he is approved by the Managing Committee of the Community.

Foundational model

The foundational model is oriented to project leaders who pretend trying to maximize the

contributions that are obtained from the community developers' collaboration.

The foundational model is based exclusively on the meritocracy for government method. Although the enormous advantages of this model have already been described in the Free Software communities' documentation, it is very important to indicate the possible disadvantages that could appear for a company that chooses this model, in principle related solely to the government of the community itself:

The project is ruled totally by the meritocracy, the leadership of the project is not fixed and it can vary democratically.

The individual developers, as they have the right to vote and representation in the project government, could become a sufficiently numerous group to become leaders of the project by their account, whenever they are organized among them to choose a leader, a leader that should be approved by commission manager of the community

Consortium Model

The consortium model is created for projects that with representation of several companies. The companies will have to reach an agreement to choose to the Director of the Project as well as the quotas of being able in the Managing Committee of the Project. Normally this agreement will be shaped in a contract. The partnership will have to present/display this contract or a document of organization where the characteristics of the project are transformed so that it is accepted by the Managing Committee, a binding agreement that will have to be respected so much by the partnership as by RobMoSys Community, although first it could change the terms of the contract whenever all the parts reach an agreement and the Managing Committee approves it.

2.3 Existing Robotic Software Platform Communities

The software industry is constantly evolving and is undergoing rapid changes. This is not only because of the evolution of technologies, but also because these technologies drive a fundamental shift in how suppliers and buyers are interrelated. It is the transition from traditional supply chains to a software ecosystem. It is not a simple linear chain anymore, but a complex network of multilateral relationships. A software ecosystem brings co-innovation as a result of different businesses interacting within a shared market for software and services, together with relationships among them. These relationships are frequently underpinned by a common technological platform and operate through the exchange of information, resources and artefacts.

Well-known examples of communities that are seen as software ecosystems are the Apple iPhone with its app store and development tools, the Google Android world, the Eclipse IDEs and, of course, the Web 2.0 technologies which turned the Internet into a platform.

The transition to a software ecosystem also comes with a shift from integration-centric approaches to a composition-oriented approach. In an integration-centric approach, the focus of an organization has been on control of the integration process. Due to the high complexity of nowadays systems, this consequently leads to unacceptable coordination costs - not to speak of the challenges of cross-department or cross-company coordination. In contrast, the composition-oriented approach does away with most of the central mechanisms and relies on a number of principles such as e.g. (i) customers compose their products by selecting from the available functionality and (ii) components satisfy the independent deployment principles.

Most popular robotic communities include ROS, OROCOS and a domain-specific community for neuro robotics is HBP, which are discussed here.

ROS

ROS (Robot Operating System) is a collection of tools, libraries, and conventions that aim to simplify

the task of creating complex and robust robot behavior across a wide variety of robotic platforms. ROS gained tremendous acceptance within the robotics community over the past years, and it has been the first time in robotics software that so many adopted a particular framework.

ROS provides hardware abstraction and device drivers, services such as message-passing between processes (nodes) and tools such as package management and data visualizers. It comes with libraries that implement commonly-used functionality in sensing, planning and control. ROS does not provide anything comparable to an IDE which could support the design, implementation and integration of ROS nodes. Instead, any preferred general purpose IDE (e.g. QtCreator) can be used. Design and implementation agreements are solely the responsibility of the user and are not tool-supported. Some user-driven initiatives tried to address this, e.g. RIDE and rxDeveloper. RIDE aims to "make the creation of ROS controllers from reusable components as easy as possible". Yet, the only provided functionality is to launch and to connect ROS nodes. Similar, rxDeveloper provides a GUI for modifying launch-file parameters of running ROS nodes. This helps in executing the nodes, but neither in their design nor implementation.

ROS is an example for *freedom of choice*. What the ROS founders mean by "we do not wrap your main" is that, among others, they do not want to enforce any architectural design decisions for developers using ROS. In consequence, each developer uses his own personally preferred architecture which is then very likely in conflict with those defined by others. This can lead to confusion as everyone first needs to understand the architectural decisions of each individual component before being able to reuse them in an own system. A proposed solution is just to extensively document each ROS component on the ROS portal. However, this does not circumvent the need to extensively analyse and understand the source code in order to adjust it or to implement workarounds in order to somehow make components compatible and reusable.

ROS, in line with its overall design philosophy, does not yet give enough structure in an appropriate format in order to better support separation of roles and separation of concerns. The minimally required structures are a sound software component model which has to be formalized for use in model-driven tools in order to support separation of concerns (e.g. to maintain semantics independently of the OS/middleware mapping), to assist the different roles in conforming to structures like component life-cycles and to reduce exposed complexity by systematic and computer-assisted management of variation points.

OROCOS

Orocos-RTT is specifically oriented towards programming and executing component-based applications on top of Real-Time Operating Systems (RTOS) and relies on lock-free communication to guarantee a deterministic execution time for all in-process inter-component data exchange. OROCOS offers a collection of commonly used components in robotic applications through a library called OCL (Orocos Component Library). For instance, we can find dedicated components to devices, hardware platforms, motion control and deployment. OROCOS allows also the integration of user-defined types through what is called typekits to ensure data transfer between processes or over a network connection. In OROCOS, there is no guidance about the control architecture to choose. The user is responsible to make his own right choices. Like ROS, OROCOS has no dedicated IDE but many initiatives like BRIDE4, RobotML5 and oroGen6 provide code generators from their models to OROCOS. They allow automatic code generation of component structures, their properties and the communication between them while having a graphical visualization about the system architecture.

Neuro Robotics Platform (HBP)

HBP is developing a novel strategy for advancing multi-level understanding of the brain, by studying brain models in the context of realistic sensory inputs and producing realistic (motor) output that

can be compared to experimental data. This is achieved by connecting the brain models to realistic virtual (or physical) bodies that are immersed in realistic dynamic environments.

In brain modelling to date, there are two complementary approaches. The first develops top-down or hypothesis driven models that focus on the functional properties of nervous systems. They define control architectures and neural network models, possibly trained by deep learning algorithms, with the aim of solving a particular set of tasks. Examples are the Spaun model (*Eliasmith et al 2012*) and control architectures commonly found in cognitive robotics. The second approach is pioneered by the HBP and consists of digitally reconstructing and simulating neural circuits or even entire brains of mice, rats and ultimately humans, based on experimental data. These bottom-up digital reconstructions focus foremost on the structural and dynamical details of the reconstructed system and regard brain function as an emergent phenomenon.

While many researchers argue in favour of one or the other position, SP10 proposes that the most productive route is to combine the two approaches: For example, many theories exist for higher-level brain functions like visual perception, but not all of these theories can be true at the same time. Some may be appropriate for humans, whereas others may be applicable to cats or rodents. The only way to separate suitable theories from less suitable ones is to give researchers a tool that allows them to confront a given theory of brain function with the anatomical and physiological realities of a particular brain embedded in a concrete body, be it mouse, cat, or human. The Neurorobotics Platform (NRP) aims to be such a tool, following the time-tested approach of analysis by synthesis.

The NRP is a powerful integration of models, simulation tools, visualisation environments and hardware-/software-in-the-loop facilities that allows neuroscientists and roboticists to connect brain models of different complexity to biological or technical robot bodies, real or virtual, that operate in complex virtual dynamic environments. The NRP is the only platform worldwide, which aims at building, operating and monitoring virtual robots of arbitrary complexity and making these models easily accessible both to neuroscientists and roboticists. It will also enable them to find "common ground" over using those robots together in simulated (or partly or fully real) environments, i.e., a basis for the exchange of ideas and concepts. To this date, such common ground hardly exists.

Since the start of the HBP in 2013, a number of commercial and open competitors have emerged. This confirms the validity and the importance of our approach. OpenAI Gym is an online platform to train top-down models in (very simple) virtual environments. A start-up in Barcelona developed a platform similar to the NRP with regard to our robot programming features, called *ROS Development Studio* (http://www.theconstructsim.com/rds-ros-development-studio/) to teach ROS. NVIDIA is offering a commercial robot simulation platform, called Isaac (https://www.nvidia.com/en-us/deep-learning-ai/industries/robotics/), which is heavily slanted towards Deep Learning. Another project that shares NRP's vision to connect brain (or AI) models to agents in dynamic environments was also launched by Google Deep Mind (https://deepmind.com/blog/deepmind-and-blizzard-open-starcraft-ii-ai-research-environment/). However, it uses computer games rather than realistic robot and environment models.

In its scope and ambition, the HBP Neurorobotics Platform is still ahead of its competitors in scope and ambition. No other platform offers such a unique combination of realistic physics based robot simulations and multi-scale neural network modelling. Only the NRP is collaborative, open access and open source. The integration of the NRP into the HBP Collaboratory gives all users access to an unprecedented amount of data and models that can be used in neurorobotics experiments as well as access to supercomputing resources in Europe.

3 Sustainability Strategy

3.1 Principles

RobMoSys aims at building an open sustainable ecosystem around public, royalty-free and implementation-driven software platform standards that will ease the development of new robotic applications in multiple domains. The innovation required to advance RobMoSys methodology and tools needs to be driven by the key industrial companies. This is why RobMoSys partners believe that it needs to join an existing industrial platform that guarantees a suitable environment for open innovation and industrial feedback.

More concretely, there is a set of specific goals to be met by the targeted community:

- **Open Innovation**: Ensuring the highest levels of productivity, reliability, service, and performance implies a continuous effort of research and development in robotics software tools. To cover the largest possible spectrum of stakeholder groups, we envisage to keep a high degree of openness. Methodological specifications will be open, not just in the sense of a dissemination with non-proprietary intellectual property rights, but also in the sense of the human development process required to reach such specifications.
- **Industry-friendly Business**. As for implementations, we envisage co-existence of both open-source and proprietary solutions. Obviously, a project with public funding and with an industrial focus, will push development of industry-grade open-source results. Special attention to open-source (with so-called "industry-friendly" licenses) is needed to make the approach accessible and to spread it easily in the community and in the market.
- **Rich Tool Ecosystem**: The numerous and complex operations required to develop and maintain robotics systems imply a high level of automation based on mature software tools. RobMoSys must be connected with a solid ecosystem of software tools providing support beyond RobMoSys features. This is why, RobMoSys target "**harmonisation**" of critical aspects of the various established digital platforms (such as OPC/UA, ROS, etc.), in particular the interactions between sub-systems, as an essential enabler.
- **Long Term Support**: The tool chain needs to remain operational for the life cycle of the robotics products; many domains need more than 10 years, and some need up to 30 years. Firstly, we will enable "**growth**" into a bigger ecosystem by helping existing players to expand through our dissemination and networking activities, and by lowering entry barriers for new players through productising core functionality, thus growing the overall market. Secondly, "**scalability**" means solving bigger problems with equal or reduced costs, which we achieve, on the one hand, through improved tool-chains, and on the other hand through model-driven predictable integration. This latter aspect is squarely aimed at the complexity-from-diversity problem mentioned in the motivation.
- **Certification & Tool Qualification**: The development of robotics systems must comply with standards and regulations impacting both the final product and the development process (certification) and tools used to build them (tool qualification).

RobMoSys partners expect that the project solutions will evolve in pace with the more challenging requirements of modern robotics engineering teams. RobMoSys solutions needs flexible extensibility and customization that makes it easier to adopt its tools, methods and processes by robotics engineering teams.

3.2 Role of the Community Platform

The goals of the potential community platform must be:

- Providing means of collaboration between end user companies.
- Organizing sustainable commercial services and ecosystems around open source models,

software components and tools.
- Fostering exchanges between academics and industry partners.
- Managing the quality and maturity of tools and components from early research prototypes through to obsolescence.
- Providing the documents and qualification kits required for certification.
- Recognizing project maturity and company know-how and commitment through a branding process.
- Ensure long-term longevity of models, software components, and tools since they must last for a long time.

The idea of joining an existing community is to avoid re-developing community management aspects but also to connect with other software communities. A lot of very good solutions answering some industrial needs already exist in open source. But most of the time, specific issues like durability or certification are not taken into account. In this case, the community platform must play its part by providing tool components assets, setting up specific support, and coordinating development and support.

3.3 Role of RobMoSys Contributors

The role of RobMoSys contributors in such a community would be:

- Preparing RobMoSys models, software components and tools to be released/hosted in open source.
- Operating dedicated code repositories, build chains, test facilities, etc.
- Fostering exchanges between RobMoSys partners and industry partners.
- Proposing RobMoSys tool enhancements (industry-friendly functionalities, new features, reliability features, tool connectors with other external tools, among others).
- Managing the quality and maturity of RobMoSys tools
- Ensuring open innovation through the sharing of the research, development, and maintenance efforts as far as possible
- Fostering sustainable commercial services and ecosystems around the RobMoSys tools.

The main goal is to use RobMoSys partner's technical expertise in the models, software components and tool platform and comprehensive understanding of the ecosystem challenges, in an effective way by offering continuous support for industrial players wishing to use these technologies in a cost-effective way for long-term projects.

3.4 Industrial Outreach

The creation of a sustainable ecosystem is a major objective of the project, but obviously, this is not something that can be done in a guaranteed or strictly controlled way. However, the more clever, more open and more reactive the strategy is, the better this goal is to reach. Anyway, the project partners though will very actively stimulate active involvement of, and controversial discussions with, a selected variety of different stakeholder groups outside of the consortium of the project representing both technology-pull (application domains) as well as technology push (technical capabilities).

This involvement (and thus commitment) of relevant stakeholder groups is "secured" by both, Tier-1 Experts Workshops and by Third-Party Funding.

Tier-1 Experts Workshops will allow to systematically gather requirements and recommendations helping project team members to (i) define the specifications of the Open Call, (ii) to monitor and assess the results of the Open Calls and (iii) to promote strategies for dissemination and exploitation. Hopefully, they will also lead to some co-development of software. Workshops during the monitoring phase of Open Calls will also host trial sessions with selected groups of users

moderated by experts.

Third-Party Funding allows the “influx of knowledge and brains” from a broad set of stakeholders, so that the project will be able to tap the tremendous potential of the community in a systematic way. In this way, RobMoSys will be able to successfully “address the open development of integrated sets of tool chains and building block applications that will support the construction of complex robotics systems”, which is a major expectation in this Call. The Open Call mechanism allows to identify the best tools already available, the best modelers and developers to adjust them and the best application areas to validate the results and establish benchmarks. This will result in standards to describe robot systems and system building blocks as well as their interaction. The resulting software architecture will be modular, composable, re-usable and easy to use. Cascade funding will also solve another burning issue: the access to integrated sets of common tool chains and real-world test installations to support the development of complex robotics systems.

"One-on-One dissemination", in which some core partners of the project engage in a co-development effort with selected individual industrial (non-funded) third parties, to help the company's engineers to adopt the RobMoSys approach and models and code in the context of that company. The motivations behind this means are (i) less risk for the adopting company, (ii) faster and more effective dissemination with more direct results, and (iii) more intense feedback for the RobMoSys partner from industrial reality.

Beyond the project duration. The change of industrial practices is a complex and long process. A single research project, like RobMoSys, cannot realistically expect to cause a revolution in industrial robotics processes in a short-term. However, the project is demonstrating a feasibility of the advocated approach and industrial RobMoSys partners, Tier-1 experts and third-party partners will be committed to internal dissemination of the outcomes and gradual adoption of both individual concepts and the entire framework developed by the project.

It should be noted that one of the secondary, but very important, by-products of the project is establishing of networks between industrial and research, training and consultancy stakeholders. Combined with exposure of industrial partners to cutting edge concepts and principles this will naturally facilitate future smaller-scale collaboration that will, in turn, facilitate adoption of RobMoSys concepts in industrial practices.

The long-term impact of RobMoSys in terms of influencing industrial practices can be facilitated by a follow-up adoption program. This will be elaborated in future versions of this deliverable (D7.3).

4 RobMoSys Community Governance

From the early stages of RobMoSys, a strong candidate for the project's open-source community was the **Eclipse ecosystem** and in particular its **PolarSys platform**, targeting tools for embedded system development. In this section, we evaluate the PolarSys platform and its governance framework, as a preliminary baseline to decide if RobMoSys fits well in this community.

4.1 Governance Models

A solution for a multiple participant, multiple product platform, such as RobMoSys, should be carefully chosen. The RobMoSys ecosystem should be segmented and for each sub-project a governance model should be written.

If we consider the Eclipse/Polarsys community, RobMoSys would need to be aligned to the Polarsys Charter (please see the Polarsys Working Group Charter[1]).

The rules of engagement of the open-source Eclipse model are based on three principles:

- **Open**: Eclipse is open to all; Eclipse provides the same opportunity to all. Everyone participates with the same rules; there are no rules to exclude any potential contributors which include, of course, direct competitors in the marketplace.
- **Transparent**: Project discussions, minutes, deliberations, project plans, plans for new features, and other artifacts are open, public, and easily accessible.
- **Meritocracy**: Eclipse is a meritocracy. The more you contribute the more responsibility you will earn. Leadership roles in Eclipse are also merit-based and earned by peer acclaim.

Meritocratic projects often start life with a small number of decision makers (at the beginning some few organizations could be the RobMoSys core partner in Polarsys). Possibly only a single person in charge of designing or distribution the decision control. However, even having a community with a minimal structure, it will ensure that the members controlling the strategy in the present will be ones in charge of identifying the future contributors for the community. This way they ensure a common vision for the project and a long term strategy. Decision-making responsibilities are usually reserved for those willing and able to understand and appropriately represent the views of the wider community. The RobMoSys ecosystem will then move to a fully flat structure (i.e. one in which decision control is distributed among community members) in recognition of the project's contribution (meritocratic governance).

Polarsys governance is based on the following principles:

- Polarsys is a user-driven organization,
- A means to foster a vibrant and sustainable ecosystem of tool components and service providers,
- A means to organize the community of each project or tool component so that users and developers define the roadmap collaboratively.

Regarding RobMoSys within Polarsys, some processes that are inherent to RobMoSys evolution will be adopted and put them into practices so as to run effectively the Polarsys community after the project ends. Some of these processes will be the communication mechanism such as the mailing list, forum/discussion guidelines, wiki contribution guidelines. Other important processes to define will be coding and releasing guidelines, community decision-taking processes and voting processes. These processes are supported by Polarsys and regulated in the Polarsys Working Group Charter.

[1] : https://www.eclipse.org/org/workinggroups/polarsys_charter.php

4.2 Governance Principles and Rules

In order to participate in Polarsys, one entity must be at least a Solutions Member of the Eclipse Foundation, have executed the IWG Participation Agreement, and adhere to the requirements set forth in the Polarsys Working Group Charter.

In order to propose RobMoSys tools and methods to be part of Polarsys, an entity must be a Committer Member. Committer members are individuals who through a process of meritocracy defined by the Eclipse Development Process are able to contribute and commit code to Polarsys projects for which they are responsible. Committers may be members by virtue of working for a member organization, or may choose to complete the membership process independently.

All Polarsys Members must be parties to the Eclipse Membership Agreement. In the event of any conflict between the terms set forth in this Polarsys Industry Working Group Charter and the Eclipse Foundation Bylaws, Membership Agreement, Eclipse Development Process, Eclipse Industry Working Group Process, or any policies of the Eclipse Foundation, the terms of the Eclipse Foundation Bylaws, Membership Agreement, process, or policy shall take precedence.

The Intellectual Property Policy of the Eclipse Foundation will apply to all Polarsys activities. Polarsys will follow the Eclipse Foundation's IP due diligence process in order to provide clean open source software released under EPL or any other licenses approved by the IWG and the Eclipse Foundation Board of Directors, such as BSD-like and LGPL.

The Eclipse Foundation Development Process will apply to all Polarsys open source projects. In particular, the project lifecycle model and review process will be followed by all Polarsys open source projects.

4.3 Collaboration Cases

In this section the different collaboration cases that might occur in the RobMoSys Community will be presented.

4.3.1 Develop

In this area we have find different cases that have the aim of making the end application more efficiency and capable, in a sum, a better application for end users.

Request new feature: Anyone (end users, experienced users, developers, supporters) can request a new feature that would help the application. Make good suggestions to enhance the system. The feature could be for anything: security, new functions, architecture efficiency, user interface ... This request would be stored and will follow the developing process.

Program new features: In this case developers accepted by the community or technical users can develop a feature and submit the implementation to the community where it will follow the acceptance process.

Port the application to other architecture: Sometimes it is not about new features but making the application available for other architectures different form the original one. To do this the developer or developers' team should be experts on the new architecture at the same time that they know the application.

Package the applications: A package is a discrete collection of web pages, code, and database tables and procedures. A developer is in charge of this task, he should follow the Maintainer Process. It is usual that the developers that code the functions of a package, maintain the package and help newcomers that are interested in that package.

4.3.2 Quality

Report a bug: Any users could find a bug and should report the finding bugs to do this, users only need to give information about what has happen with the application when they find the bug. Reporting the bug is a way of performing the application. Anyway, before reporting a bug, the user should assure that the bug has not been previously reported.

Complete the information about bugs: This collaboration case is related to the previous one. Many of the bugs are reported by end users that hardly ever have a technical background. The collaborator should first assure that it is really a bug and then complete the information, especially with the test cases that would help to probe the bug solution.

Solve a bug: The bugs are there, no matter they are reported, they will be there until someone solve them. Collaborator can solve a reported bug or even solve a new bug that hasn't been reported yet. The solution implementation should follow it

Debug existing features: Before a new release come to general public, not only the new features have to be implemented but they also must follow a testing period where tester user should use the application and probe the efficiency of the new features and the integration with other parts of the application.

Add documentation: The collaborator required technical background and should have knowledge of the application. This collaboration case will help new developers to understand the application and make them be ready to develop code for the community.

4.3.3 Help

Write tutorial: Experienced users could help newcomers to get in touch with the application for the first time or explain how the new features work.

Help users: In open source community there are different communication channels that end users use to get information about problems they have with the application. In this case the collaborator should use at least one of the communication channels (distribution lists, IRC channel, forums...) to help users with their problems. Each channel has a different way to interact so the collaborator should chose the channels he is comfortable on.

Maintain the application: Collaborator with technical background could help the community by spotting abandoned/poorly maintained packages and join the maintenance teams. He can also help triaging bugs related to the application or simplified the procedure to maintain packages.

4.3.4 Internationalization

Translate the applications: The collaborator should write fluently the language he wants to translate and it is desirable to be an expert user. He should join existing localization teams so he can help the existing translator or create a new team if the existing teams have not work on the selected language. He should work as part of team translating or/and revising the work of others.

Translate the documentation. This collaboration case it is often ignored, but is as important as the one before. The collaborator should keep inform of the work done by teams in change of writing documentation or tutorials. It is desirable to have technical knowledge as the technical documentation is hard to translate at the same time as the technical concepts are remain the same without the technical understanding.

4.3.5 Support and Dissemination

Help development the application public face: To make the project known by others is important, that way the community start growing and so does the development with the help of new

developers. To make the community and the project known, the community need collaboration to: create and maintain the web page, assist to events and conferences where collaborators should disseminate the objectives and goals of the community...

Donate equipment and services: The community infrastructure needs donations so it will be available to everyone, no matter where. As the community grown the needs for own infrastructure are bigger and it's more difficult to find free infrastructure. Organizations or individuals who want to support the work could donate equipment or services like hosting, legal consultancy.

Promote the project: Everyone could promote the project. Collaborator should invite others to get to know the community and what it offers, a short of evangelist. Anyone could make a donation to help managing the community and so as to pay for not free services. That donation doesn't have to be always money, it could also be time. The community apart from develops need people with other skills: legal knowledge to work with licences, carpenters to help with the stands in events.

4.3.6 Supplementing the License--Contributor Agreements

Many open-source projects require that developers wishing to contribute to the project sign an additional form. Developers must state that they have the right to contribute their source code--that is, that they own it and that it does not belong to someone else. If the developer works for a company, the form has the developer state that the company grants the right to use the contributed code to the project. Note that even developers doing open-source work on their own time might still need to get approval from their employers because employee agreements often specify that the company owns anything employees invent.

The form also usually states that the developer grants to the project the right to freely use any patents and third-party IP used by the contributed code.

Some projects use the form to assign the copyright of the code to the project. Other projects, such as Open Office, use a joint copyright assignment so that both the project and the contributor retain full rights to use, modify, and redistribute the copyrighted work.

Developers usually need to sign the form only the first time they contribute code; the form then applies to any subsequent contributions. Developers are still free to exercise their rights to the code, but, if they want to participate with the specific project code base, they must sign the form.

If the additional requirements of the contributor agreement form are not acceptable, then a developer may be motivated to join or create a new project, that is, to fork the project.

4.4 Decision Making Process

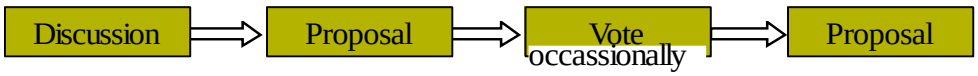
In an open source community as the one that RobMoSys is aiming to generate, how decisions are taken must be clarified. This point describes different mechanisms for this decision making processes.

Decisions about the future of the project are made through discussion with all members of the community, from the newest user to the most experienced PMC (see Section 3, Roles section) member. All project management discussion takes place on the project-contributors mailing list (occasionally sensitive discussion occurs on a private list, but general project management discussion always occurs on the public lists).

These are the different activities that describe the process of the decision making.

1.1.1 Lazy Consensus

Decision making is usually a two-step process. Occasionally a third step is required. These steps are:



Any community member can table an idea for consideration. In order to prompt a discussion about a new idea, an email is sent to the project contributors. Once the idea has become more concrete and appears to have a significant level of support within the community it is time to make a proposal.

Usually, gaining consensus within the community is easy since all community members have a common set of goals. By the time an idea reaches the proposal stage it is likely to have community support since it will already have been discussed publicly. However, full community consensus cannot be assumed at this stage, hence the need for a proposal stage.

In general, as long as nobody explicitly opposes a proposal then it is recognized as having the support of the community. This is called lazy consensus, that is, those who have not stated their opinion explicitly have implicitly agreed to the proposal being implemented. The "[Proposal]" subject tag is used to grab the attention of community members who may have been unable to participate in the discussion phase. It is an indicator that a project decision is about to be made. It is this that allows the community to assume that lazy consensus has been reached.

Lazy consensus is a very important concept within the project. It this process that allows a large group of people to efficiently reach consensus. This efficiency stems from the fact that someone with no objections to a proposal need not spend time stating their position; furthermore others need not spend time reading such mails.

The time spent gaining the consensus of the community does not prevent work proceeding on the idea. People are free to work on the implementation of any idea or proposal whilst the community examine and discuss it. This process need only be followed for an action that will significantly affect the project. Smaller actions, such as the addition of an optional feature or the fixing of a reported bug need no discussion. Contributors are free to action these items. Since this process is only followed for major activities the time spent reaching consensus is usually considerably less than the time spent implementing the decision.

4.4.1 Voting

Not all decisions can be made using lazy consensus. Issues such as those that affect the strategic direction of the project must gain explicit approval in the form of a vote. See the next section for a discussion of when a vote is needed.

In the case of Eclipse/PolarSys, a **super majority** approach exists: for actions (i) requesting that the Eclipse Foundation Board of Directors approve an additional distribution license for Polarsys projects; (ii) amending the terms of the Polarsys Participation Agreement; (iii) approving or changing the name of Polarsys; (iv) approving changes to annual member contribution requirements; any such actions must be approved by no less than two-thirds (2/3) of the representatives in good standing represented at a Steering Committee meeting at which a quorum is present.

4.5 Committees

If we select the meritocracy governance model, some typical roles should exist:

4.5.1 Users

Users are community members who have a need for the project. Anyone can be a user, there are no special requirements. Users are the most important members of our community, without them the project is nothing.

4.5.2 Contributors

Contributors are community members who contribute in concrete ways to the project. Contributions can take many forms, such as those outlined below and in the above section on users. Anyone can become a contributor. There is no expectation of commitment to the project, no specific skill requirements and no selection process.

Contributors will already be performing actions as a user (see above) but will also find themselves doing one or more of the following:

- supporting new users (users are often the best people to support new users)
- reporting bugs
- identifying requirements
- Graphics and web design
- programming
- assisting with project infrastructure
- writing documentation
- fixing bugs
- adding features

As contributors gain experience and familiarity with the project they may find that making such contributions becomes easier. As a contributors profile and commitment increases they may be nominated for committer ship, as described in the next section.

4.5.3 Committers

Committers are community members who have shown that they are committed to the continued development of through the project ongoing engagement with the project and its community. Committer ship allows contributors to more easily carry on with their project related activities by giving them direct access to the projects resources. That is, they can make changes directly to project outputs rather than having to submit changes via patches.

This does not mean that a committer is free to do what they want. Actually they have no more authority over the project than a contributor (although they do have a "binding vote", see the section on Decision Making below). Instead it means that they have shown themselves to be valued members of the community with a healthy respect for the projects aims and objectives. Their work continues to be reviewed by the community and it must be accepted by the community before it will be included in a release. However, this approval is sought after the contribution is made, rather than before.

This is known as a commit-then-review process. It is more efficient to allow trusted people to make direct contributions as the majority of those contributions will be accepted by the project. We employ various communication mechanisms to ensure that all contributions are reviewed by the community as a whole; however, there is no need to detail them here. By the time you are invited to become a committer you will have been guided through the use of our various tools as a user and then contributor.

Anyone can become a committer; there are no special requirements other than to have shown they are willing and able by participating as a contributor. Typically a committer will need to show they have an understanding for the project, its objectives and its strategy. They will also have provided valuable contributions to the project over a period of time.

New committers can be nominated by any existing committer. Once nominated there will be a vote by the Project Management Committee (see below). Committer voting is one of the few activities that happen on the private management list of the project. The reason it occurs in private is to allow Project Management Committee members to freely express their opinions about a nominee

without having to do so in public where it may embarrass someone. Once the vote has been held the aggregated voting results are published on the public mailing list. The nominee is entitled to request an explanation for any "no" votes against them regardless of the outcome of the vote. This explanation will be provided by the Project Management Committee Chair (see below) and will be anonymous and constructive in nature.

Nominees may refuse their appointment as a committer. However, to do so is unusual as the project does not expect any specific time or resource commitment from its community members. The role of committer is intended to allow people to contribute to the project more easily; it is not intended to tie them into the project in any formal way.

A committer who shows an above average level of contribution to the project, particularly with respect to the strategic direction and long term health of the project may be nominated to become a member of the Project Management Committee, this role is described below.

4.5.4 Project Management Committee

The Project Management Committee (PMC) of the project consists of those individuals identified as "Project owners" on the development site. The PMC has additional responsibilities to ensure the smooth running of the project. PMC members are expected to review code contributions, participate in strategic planning, approve changes to the governance model and manage the copyrights within the project outputs.

Being a member of the Project Management Committee does not give significant authority over other members of the community, although it is the PMC who vote on new committers and make decisions when community consensus cannot be reached. The PMC also has access to the projects private mailing list and its archives. This list is used for sensitive issues (such as votes for new committers and legal matters that cannot be discussed in public), but is never used for project management or planning.

Membership of the PMC is by invitation from the existing PMC members. A nomination will result in discussion and then a vote by the existing PMC members. PMC membership votes are subject to consensus approval of the current PMC members.

4.5.5 Project Management Committee Chair

The PMC Chair is a single individual, voted for by the PMC members. Once someone has been appointed chair they remain in that role until they choose to retire.

The PMC Chair has no additional authority over other members of the PMC. The role is one of coordinator and facilitator. The chair is expected to ensure all governance processes are adhered to.

The PMC Chair also has the casting vote in the event of an inability of the project to reach consensus or the required number of votes when a vote is called.

4.6 Community Tools

One of the key aspects for a successful RobMoSys community is the necessity of proper communication tools.

4.6.1 E-Mail distribution lists.

The most usual and used way of communication between all the members of an Open Source community is an e-list, or an electronic mail distribution list.

An electronic mailing list (sometimes written as a list or e-list) is a special usage of email that allows for widespread distribution of information to many Internet users. It is similar to a traditional

mailing list - a list of names and addresses - as might be kept by an organization for sending publications to its members or customers, but typically refers to four things: a list of email addresses, the people ("subscribers") receiving mail at those addresses, the publications (e-mail messages) sent to those addresses, and a reflector, which is a single e-mail address that, when designated as the recipient of a message, will send a copy of that message to all of the subscribers.

4.6.2 Internet forums

As they are very similar to the distribution lists, distribution lists can be mapped as forums items in a tree-like hierarchy model, they play a key role in the communication process of an Open Source community.

Originally modelled after the real-world paradigm of electronic bulletin boards of the world before Internet was born, internet forums allow users to post a "topic" for others to review. Other users can view the topic and post their own comments in a linear fashion, one after the other. Most forums are public, allowing anybody to sign up at any time.

Forums can contain many different categories in a hierarchy according to topics and subtopics.

Current successful services have combined new tools with the older newsgroup and mailing list paradigm to produce hybrids.

4.6.3 Wikis

A wiki invites all users to edit any page or to create new pages within the wiki Web site.

Wiki promotes meaningful topic associations between different pages by making page link creation almost intuitively easy and showing whether an intended target page exists or not.

A wiki is not a carefully crafted site for casual visitors. Instead, it seeks to involve the visitor in an ongoing process of creation and collaboration that constantly changes the Web site landscape.

A wiki enables documents to be written collaboratively, in a simple mark-up language using a web browser. A single page in a wiki website is referred to as a “wiki page”; whiles the entire collection of pages, which are usually well interconnected by hyperlinks, is "the wiki". A wiki is essentially a database for creating, browsing, and searching through information.

Inside an Open Source community a wiki page can be used for many different things, but the facilitation of the documentation is obvious, bug tracking, all kind of enumerations of “to-Do”'s, etc., can easily be managed by the.

4.6.4 Chat Rooms

In an Open Source like community, Chat Rooms can be used for the everyday development process, as a direct knowledge sharing procedure, deeper conversations with any of the involved participant in any related domain of the project, etc. This chat room, if used, makes participants fell closer to the project, to each other and it's very positive for the development of the project.

Or it can play a key role in a concrete moment of the development process in an efficient meeting point of as many participants as needed for talk and adopt a concrete solution for a given problem.

Internet Relay Chat (IRC) and other on-line chat technologies allow users to join chat rooms and communicate with many people at once, in a public way. Users may join a pre-existing chat room or create a chat room about any topic. Once inside, you may type messages that everyone else in the room can read, as well as respond to messages from others. Often there is a steady stream of people entering and leaving. Whether you are in another person's chat room, or one you've created yourself, you are generally free to invite others on-line to join you in that room. Instant messaging facilitates both one-to-one (communication) and many-to-many interaction.

4.6.5 Forges

Many of the communication tools that have been shown here and other for developing purposes can be recompiled in a forge. It's a way to have all the tools manage in one place and could increase the dissemination of a project. The largest open source forge is Sourceforge with more than 230,000 software projects hosted and more than 2 million users registered.

4.7 Sustainability View per Partner

4.7.1 Summary of RobMoSys partners View

All the RobMoSys partners agree on joining an existing community or foundation to maintain the RobMoSys ecosystem. It seems to be more effective to establish a working group within an existing, well established foundation that has already a reputation and is known in a certain user group.

Most of the RobMoSys partners agrees that the most suitable community would be Eclipse. Since the creation of the first Eclipse Working Groups (the IoT Working Group and PolarSys), the Eclipse Foundation has accumulated lot of experience in creating such an ecosystem with different level of collaboration on technology and on the development of the ecosystem.

It is important to make an explicit distinction between different RobMoSys assets: (i) models, (ii) software components, i.e. code that realises the models, and (iii) tools. Tools can help the transformation from (i) into (ii). It is not necessary that all three are hosted in one single community. While there is a huge advantage of putting (iii) inside existing communities like Eclipse, there are advantages to have (i) and (ii) not in the same host. This is something we must assess in the planned workshops with RobMoSys stakeholders and Tier-1 experts.

It is also important to note that it is essential to sustain RobMoSys activity in communities that are outside of the RobMoSys community. In order to make the RobMoSys activity sustainable after the end of the project it is also necessary to build a dynamic, growing community around it.

4.7.2 Questionnaire feedback

In this section, we present a summary of the sustainability views 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 sustainability model	Not yet clear and currently under discussion. At least, for the Eclipse-based SmartMDSD toolchain of HSU with its RobMoSys conformant extensions, it can make sense to relate it to the Eclipse foundation.

Partner	COMAU
Most appropriate sustainability model	Industrial (Robotics).

Most appropriate sustainability model	Working group inside an existing foundation Eclipse, Linux Foundation, etc. (direct funding, with different levels of membership fees).

Partner	ECL
Type of Organization	Open Source Foundation.
Most appropriate sustainability model	The privileged model for the Eclipse Foundation will be the creation of an Eclipse Working Group. Since the creation of the first Eclipse Working Groups (the IoT Working Group and PolarSys), the Eclipse Foundation has accumulated lot of experience in creating such an ecosystem with different level of collaboration on technology and on the development of the ecosystem. We think that this experience will help RobMoSys create a successful ecosystem around the technologies.

Partner	KUL
Type of Organization	Research & Education.
Most appropriate sustainability model	Probably, any of the above. As an university/research institution, the major form of sustainability of RobMoSys activity is with follow-up funded projects and initiatives, with the aim to improve or add models and functionalities to the RobMoSys software baseline. A possible turnover can be generated by the creation of innovative university spin-off, aiming to deliver (i) educational services, such as training of the existing tools, (ii) consultancy support to interested third-parties and (iii) development of models, functionalities and configurations for specific, advanced application scenarios. Once again, the main idea is to have models and a software baseline that supports those completely open and free-of-charge. Instead, application-dependent configurations can be developed upon third-parties requests by means of one (or more) RobMoSys "experts", under specific agreements case by case. In this way, RobMoSys "experts" (aka, RobMoSys community) are motivated to maintain and further developing the RobMoSys approach, since it makes the application design and support affordable in their business model, whether it is an university/research institution or a company. To this end, a private foundation with membership fees or any other form that prevent the models (and tools that support those) to be visible by parties not in the RobMoSys consortium will not be compliant with the above-mentioned sustainability strategy.

Partner	CEA

Type of Organization	Research & Education. CEA is a Research and Technology Organization. Two laboratories of CEA are involved in RobMoSys project: • 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 sustainability model	Sustainability of RobMoSys activity requires: • An appropriate and accessible structure to host RobMoSys shared results and allowing their evolutions; • Economically viable structure (costs sharing); • Accessibility by the different categories of the robotics community; A working group in an existing foundation (equivalent to Polarsys within the Eclipse foundation) could be a first an easy to setup sustainability model. In that context CEA LIST will contribute to the group by participating to the RobMoSys meta-models evolutions and by providing parts of its Papyrus4Robotics toolchain. It is important to note that it is essential to sustain RobMoSys activity in communities that are outside of the Eclipse community. Different natures of infrastructure like Digital Innovation Hubs could also be good candidates.

Partner	EUnited
Type of Organization	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. Our focus is pretty much on industrial robotics.
Most appropriate sustainability model	The effort to found an own foundation within the runtime of RobMoSys that can work immediately self-sufficient is quite large. To find enough members to support such a foundation from the beginning to work sustainable might even not be feasible and would negatively impact to outcome of the whole project. We would see a working group within the DIH network skeptical, as they are just about to be installed. It is a relatively new tool in the EC framework, we do not know about sustainability of the DIHs yet (What happens if after 3 years EC decides to opt for an alternative?). DIHs do not have a track record yet. It seems to be more effective to establish a working group within an existing, well established foundation that has already a reputation and is known in a certain user group. This is just our preference from the proposed option. We would not actively oppose any other of the given option, if they find a majority amongst RobMoSys Consortium members. We will do our best to support the establishment of the final solution, however it will look like. In our case, this would most likely be active promotion in the robotics community and convince companies to join and support RobMoSys after the funded period.

Partner	PAL
Type of Organization	Industrial (Robotics).
Most appropriate sustainability model	Components will be available publicly with an open source license, but only the ones that meet the quality requirements will be accessible to the membership through fees.

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 sustainability model	Working group inside an existing foundation Eclipse, Linux Foundation, etc. (direct funding, with different levels of membership fees)

Partner	TUM

Type of Organization	Research & Education.
Most appropriate sustainability model	Due to the proliferation of Digital Innovation Hubs and topic-focused platforms, it does not make sense to set up yet another network striving for self-sustainability (for instance with membership fees). There is a high risk at present that the proliferation of (uncoordinated) initiatives contribute to the fragmentation of the community (and the software users) rather than to contribute to reduce this. As RobMoSys will be cross-sectional, joining forces with other initiatives like DIH and platforms is the best approach. And the topic group can play a key role in this. On the other hand, the plan of having a foundation set up after the runtime of the project – either on its own right or as a joint venture with existing foundations seems to be a good approach for us which does not conflict with any of the initiatives undertaken on a European level right now. At the end of the project, the community may not be big enough yet to support a private foundation by its own. Therefore, it may be a better solution to start a working group inside an existing foundation (Eclipse, as the project member) and benefit from the larger community, which can partially also adopt the developments of RobMoSys. In order to make the RobMoSys activity sustainable after the end of the project it is necessary to build a dynamic, growing community around it. The experiments funded via the open calls can be a solid base to form it. However, support is needed to make the community grow and thrive. Of course, cooperation with any DIH networks will be of utmost importance and has to be a part of the responsibilities of the organization managing the community.

5 Conclusions

Deliverable D7.2 presents the first version of the Sustainability Plan for the RobMoSys ecosystem. This first iteration is focused on establishing the basics and on specifying a framework for community management and provision of activities for ensuring a strong sustainability of the RobMoSys results. This scope will be extended by using the feedback of Tier-1 experts coming from the community management world workshops through workshops organised in the upcoming years.

We presented the main principles behind the sustainability of software platforms under open source schemes, as well as the main strategic decisions to guarantee the evolution and maturity of the RobMoSys assets (models, software components and tools). This report summarises efforts to draw the interest of open-source communities in the project, by ensuring the availability of means for interaction with and collaboration between the community members through required communications means, and by providing helpfulness and support required to build and maintain the community.

From the early stages of RobMoSys, a strong candidate for the open-source community, and in particular for hosting the tool assets, has been the Eclipse ecosystem. We must assess this option and any related mechanism around this option from the second year by taking into account that the innovation required to advance RobMoSys ecosystem needs to be driven by the key industrial companies.

RobMoSys partners expect that the project solutions will evolve in pace with the more challenging requirements of modern robotics systems and will provide more flexible extensibility and customization that makes it easier to adopt the tools to the methods and processes of robotics engineering teams.

6 References

[Commission 1987] Commission, B., Our common future, Chapter 2: Towards sustainable development. World Commission on Environment and Development (WCED). Geneva: United Nation, 1987