

Towards a Common Robotics, Automation and Manufacturing Operating System (*CRAMOS*)

Herman Bruyninckx

Dept of Mechanical Engineering, K.U.Leuven, Belgium

<http://people.mech.kuleuven.be/~bruyinck/>, <http://www.orocos.org>

Version 0.12, November 11th, 2007

Abstract

This document explains a vision about (i) the goals of an ecosystem of *independent but interoperable* software projects (available under *Free Software/Open Source* (FOSS) licences) to serve as an *operating system* (a *framework*, or a *middleware*) for all software aspects in robotics, automation and manufacturing; (ii) what could be the components of such a *Common Robotics and Manufacturing Operating System* (CRAMOS); and (iii) a strategy for achieving the stated goals. Many of the goals can be reached by rather straightforward technical refactoring of existing projects, in combination with the use of existing general purpose open software standards, and the definition of new, CRAMOS-specific open standards. However, the major obstacle towards these goals is not technical, but social: the non-negligible reluctance of FOSS project developers to accept standards and to do refactoring for the sole purpose of improving interoperability with other projects without short-term benefits to their own project.

1 Introduction

Contrary to the successful and similar cases of, say, servers, High Performance Computing, mobile telephony, or consumer electronics, there is currently no professional-grade, company-friendly, “Linux-like” software ecosystem for the programming and control of all sorts of robotics, automation and manufacturing *devices* and *systems*, from the lower level “operating system” support to higher level development, utility and application programs, including extensive and consistent documentation, standardisation and packaging support. The sub-domains of robotics, automation and manufacturing are natural matches, because they all involve *planning*, *sensing* and *control* on realworld machinery, albeit at different levels of *intelligence*, *distribution* and *realtime reactivity*. In addition, modern manufacturing systems can involve various robots and automation devices. The software aspects of such systems can be divided roughly into the following levels¹:

1. **Corporate level:** to manage factory marketing and sales, accounting, product design, research, aggregate planning, master production scheduling, etc.
2. **Plant level:** to manage *all* the plant’s shop floors, i.e., groups of machine tools, robots, material handling and transport devices.
3. **Cell/shop floor level:** to manage *one single* manufacturing cell, consisting of the same types of devices as on the Plant level.
4. **Machine level:** to control the *discrete* aspects of one single device, i.e., its throughput, maintenance schedule, uptime history, etc.
5. **Device level:** to control the *continuous* aspects of one single device, i.e., the *realtime feedback control* loops around sensors and actuators.

¹The contribution of Indra Tanaya Priangada to the inclusion of this information is gratefully acknowledged.

The focus of this document is on the last three levels, for which it outlines a strategy to create a software ecosystem, which will be called *Common Robotics, Automation and Manufacturing Operating System* (CRAMOS). Its development should ideally be driven by a “portal organisation” (consisting of major FOSS projects in the CRAMOS domain) that *stimulates* developments within independent but *interoperable* software projects, instead of *doing* all the developments itself. The CRAMOS initiative could organise fundraising to fill concrete, well-identified holes in the desired ecosystem, and to coordinate integration and standardisation efforts.

2 Goals

The CRAMOS initiative has technical, commercial, *and* community goals, that should appeal to a large set of research labs, companies, service providers, educational organisations, and existing FOSS projects. The following Sections give an overview of these goals, with an emphasis on the technical goals.

2.1 Technical goals

On a technical level, the goal is the development of a multi-platform ecosystem of interoperable *FOSS* software projects that provide the infrastructure for all needs in programming, control, simulation and visualisation, and supporting all possible kinds of robotics and automation devices. More concretely, the following components are *required* (\mathcal{R}), *desired* (\mathcal{D}), or *nice to have* (\mathcal{N}), but the mentioned *sources of inspiration* are often still not mature enough to be useful in the professional-grade software ecosystem this document is aiming at:

\mathcal{R} *Hardware Abstraction Middleware*, containing device drivers for all possible robots, automation devices, communication buses, and operating systems, with a standardized interface to CRAMOS. The standardization is very important, in order to facilitate, both, the integration of new robotic devices into the CRAMOS ecosystem, and the immediate availability of new hardware to all interested projects.

Sources of inspiration: HAL (<http://www.freedesktop.org/wiki/Software/hal>), D-BUS (<http://www.freedesktop.org/wiki/IntroductionToDBus>), Real-Time Driver Model (<http://www.xenomai.org/documentation/branches/v2.3.x/pdf/RTDM-and-Applications.pdf>), Comedi (<http://www.comedi.org>).

\mathcal{R} *Ontologies*, to document the meaning of all terms, concepts and APIs (Application Programming Interfaces) used in the CRAMOS software. The experience of several decades of large-scale technical software projects is that too many “implicit assumptions” and “hidden policy choices” slip into the code at various places, leading to correctly implemented but nevertheless incompatible libraries, because the exact meaning of all terms and the exact behaviour of all components in the common interfaces has not been formalized explicitly or completely.

The simplest ontology would be something like an extensive *glossary* of terminology; the complete version would have all definitions available in computer-interpretable form.

Sources of inspiration: <http://www.planetmath.org>, <http://en.wikipedia.org/wiki/robotics>.

\mathcal{R} *Standards*, to allow interoperability between different projects. Standards must be developed at at least four complementary levels: symbolic representation (“ontology”, “semantics”), mathematical models, computer representation, and hardware representation. It’s better to develop a family of small standards with well-defined scope, instead of one huge, monolithic API.

Sources of inspiration: RoSta (<http://www.robot-standards.org>), <http://people.mech.kuleuven.be/~brwyninc/robotstandards/>.

\mathcal{R} *Communication middleware* to facilitate the network-transparent exchange of data and “events” between CRAMOS software modules.

The interpretation of *middleware* (<http://en.wikipedia.org/wiki/Middleware>) changes over time: the more application functionalities become “commodity”, the higher the middleware software stack can grow, providing transparent access to all these functionalities. (The concepts of “middleware”, “framework” and “operating system” are considered to be synonyms in this document.)

Sources of inspiration: *Gstreamer* (<http://gstreamer.freedesktop.org>); *EPICS* (<http://www.aps.anl.gov/epics>); *Clam* (<http://clam.iua.upf.edu>); *Miro* (<http://www.informatik.uni-ulm.de/neuro/321.html>) and *URBI* (<http://www.gostai.com>) and *Orca* (<http://orca-robotics.sourceforge.net>), some

of which rely on lower-level middleware such as *Ice* (<http://www.zeroc.com>) and *ACE/TAO* (<http://www.cs.wustl.edu/~schmidt/ACE.html>).

\mathcal{R} *Realtime control framework* to facilitate the writing of interpolation, control and estimation loops in hard real-time. The framework should have “thread-safe”, operating system independent and efficient support for *Supervisory Control And Data Acquisition* (SCADA, <http://en.wikipedia.org/wiki/SCADA>), *performance instrumentation* (timing, resource usage, event logging, ...) and *data logging* (http://en.wikipedia.org/wiki/Data_logging).

Sources of inspiration: *Open Robot Control Software* (Orocos, <http://www.oroocos.org>), *Beremiz* (<http://www.beremiz.org>), *EPICS* (<http://www.aps.anl.gov/epics>); *MatPLC* (<http://www.matplc.org>), *Enhanced Machine Controller* (<http://www.linuxcnc.org>).

\mathcal{R} *Non-realtime activity control framework* to support the monitoring and scheduling of “order processing” in a manufacturing cell, a job shop, or even a factory.

Sources of inspiration: *TinyERP* (<http://www.tinyerp.org>).

\mathcal{R} *Software services and deployment framework* to keep large-scale software systems up and running continuously, while allowing “hotplugging” of new functionalities and updating of existing services.

Sources of inspiration: *OSGi* (<http://www.OSGi.org>) implementations, such as *Apache Felix* (<http://felix.apache.org>) and *Knopflerfish* (<http://www.knopflerfish.org>).

\mathcal{R} *Systems and control library* to model and simulate the behaviour and the control of general dynamics systems, including digital filtering.

Sources of inspiration: *Clam* (<http://clam.iua.upf.edu>); *SDS* control blocks (Lund University, URL not yet published).

\mathcal{R} *3D visualisation and simulation framework*, to facilitate the off-line simulation, visualisation and programming of CRAMOS devices.

Sources of inspiration: *Blender* (<http://www.blender.org>).

\mathcal{R} *Integrated Development Environment* (IDE) to support the writing of application programs. The IDE should have far-reaching tools to automatic code generation, *model-driven architecture*, domain-specific plug-ins, graphical programming, development of graphical user interfaces, etc.

Sources of inspiration: *Eclipse* (<http://www.eclipse.org>, [http://en.wikipedia.org/wiki/Eclipse_\(software\)](http://en.wikipedia.org/wiki/Eclipse_(software))), and some of its domain-specific specialisations such as *OpenEmbeDD* (<http://www.openembedd.org>).

\mathcal{R} *Human Machine Interfacing framework* to display and process the data streams from real or virtual CRAMOS devices, and to offer simple programming. The HMI should offer “widgets” that are directed to be used in (embedded and desktop) applications in the CRAMOS domain, i.e., to program and interface controlled devices.

Sources of inspiration: *Maemo* (<http://www.maemo.org>), a toolkit for embedded 2D HMI; *KST* (<http://kst.kde.org>) for advanced 2D plotting; *Clam* (<http://clam.iua.upf.edu/>); the *Game Engine* in Blender.

\mathcal{R} *Multi-agent coordination specification*: robotics, automation and manufacturing systems evolve towards more and more quite independent components or “agents”, whose activities must be coordinated to achieve the system goals. Such coordination requires a formal specification language, that provides the synchronization, data exchange, and deployment of a dynamic number of agents in the system.

Sources of inspiration: the *work flow* patterns described at <http://www.workflowpatterns.com>, and the *work flow* FOSS projects (such as *Kepler* (<http://kepler-project.org/>) and *Ptolemy II* (<http://ptolemy.eecs.berkeley.edu/ptolemyII/>)), and standards (such as *WSBPEL* (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel)).

\mathcal{D} *Packaging*, to make all software available to new users in a simple-to-install package, with tutorials and examples.

Sources of inspiration: all Linux distributions.

- \mathcal{D} *Kinematics and dynamics library* to model and simulate any given *kinematic chain*, from simple mobile platforms and serial robot arms, to multi-loop devices with friction, elasticity and significant dynamics.
Sources of inspiration: *Kinematics and Dynamics Library* (KDL, <http://www.orocos.org/kdl>), *OpenSim* (<https://simtk.org/home/opensim>), *RobWork* (<http://www.mip.sdu.dk/robwork/>).
- \mathcal{D} *Physics engine*, that can model and simulate the physical interaction between any given set of (non)rigid objects.
Sources of inspiration: *Open Dynamics Engine* (ODE, <http://www.ode.org>), *Bullet* (<http://www.continuousphysics.com/Bullet/>).
With a little bit more effort (because the underlying DAE solvers are the same) one could work towards a *multi-domain physics library*. For example, *openModelica* (<http://www.ida.liu.se/labs/pelab/modelica/OpenModelica.html>) built on top of the *Modelica* modelling language (<http://www.modelica.org/>).
- \mathcal{D} *Bayesian network library* to model and simulate any estimation and learning system.
Sources of inspiration: *Bayesian Filtering Library* (<http://www.orocos.org/bfl>), *Bayes++* (<http://bayesclasses.sourceforge.net/Bayes++.html>).
- \mathcal{D} *Planning library* to plan device actions, from simple 2D geometric motions, over sequences of automation tasks, to dynamic interactions between humanoid robots and humans.
Sources of inspiration: none at the moment.
- \mathcal{D} *Computer vision library* to provide algorithms for processing camera images.
Sources of inspiration: VXL (<http://vxl.sourceforge.net/>), OpenCV (<http://opencvlibrary.sourceforge.net/>).
- \mathcal{D} *Database and binary formats* to store large amounts of experimental data, to offer advanced query and data mining access to those data, and to interface with real and simulated devices.
Sources of inspiration: Round Robin database (<http://torrus.org/>); C3D (<http://www.c3d.org/>); netCDF (<http://www.unidata.ucar.edu/software/netcdf/>); HDF5 (<http://hdf.ncsa.uiuc.edu/products/hdf5/index.html>).
- \mathcal{D} *Mathematical prototyping tool* (à la Matlab/Simulink), with extensive robotics-related algorithms, and automatic code generation.
Sources of inspiration: Octave (<http://www.octave.org/>), Scilab (<http://www.scilab.org/>), SciPy (<http://www.scipy.org/>).
- \mathcal{D} *Benchmark repository*, to compare the relative and absolute performance of algorithms.
- \mathcal{N} *Unit testing framework*, http://en.wikipedia.org/wiki/Unit_test.
Sources of inspiration: Salome (<https://wiki.objectweb.org/salome-tmf>).
- \mathcal{N} *On-line robotics textbooks*, to describe the technical and technological aspects of robotics in detail.
Sources of inspiration: the *Robotics WEBook* (<http://www.roble.info/>), *Wikipedia* (http://en.wikipedia.org/wiki/List_of_basic_robotics_topics).

2.2 Commercial goals

The (nicely packaged) combination of (most) above-mentioned projects would provide a common, pre-competitive, continuously maintained software infrastructure of high quality, with company-friendly FOSS licenses. Companies can add their own *added value* to this infrastructure, without having to pay for all of the development and maintenance of that infrastructure but at the same time safeguarding their added value intellectual property. The most mature “business models” in this context are those of the (i) Linux kernel, (ii) the Linux distributions, and (iii) the independent software vendors (ISV):

- *Linux kernel*: CRAMOS should be managed by something like the *Linux foundation* (<http://www.linux-foundation.org/>), i.e., one single legal entity that defines what CRAMOS is, and that can hire developers, because it gets funded by companies that want to outsource the infrastructure developments.

The important difference between Linux and CRAMOS is that Linux has a commonly accepted “dictator” in the person of [Linux Torvalds](#), while the robotics community lacks such a person or organisation. So, the creation of *one* centralized “CRAMOS foundation” could be problematic, because it can have problems finding the unanimous support of all parties involved.

There exists already a German initiative in this direction: the [Open Source Automation Development Lab](#) (OSADL). Its current focus is on improving the realtime status of the Linux kernel, and on providing device drivers.

- *Linux distributions*: three or four companies could be created, that make money by *packaging services* around the free platform. Each company would typically target one specific CRAMOS domain: industrial automation; mobile robotic platforms; manufacturing; medical applications; hobbyists; etc.
- *ISV*: several companies can make money by *application customization services* on top of the FOSS platform.

The above-mentioned OSADL organisation already has such ISV companies among its members.

There is room for companies in all three of these business models, but these companies will (because of the nature of FOSS) be small and operating worldwide. In addition, it might be appropriate for them to be (or to become) part of a larger (FOSS) company.

Alternative to *Microsoft Robotics Studio*. CRAMOS should also be marketed as *the* alternative to *Microsoft Robotics Studio*, as the “Linux/Eclipse” duo is considered as the only viable long-term alternative to the WinCE/Visual Studio duo of Microsoft. The added value of CRAMOS lies in the same aspects as in the Linux/Eclipse case:

- it targets more than robotics only: also automation and manufacturing are part of its target domains.
- it’s FOSS. Hence, it is easy to extend and to customize.
- it will *not* be *single-vendor*.
- it will be really *multi-platform*, i.e., run under Windows, Linux, Mac OS X,
- it will not be *one size fits all*, but different libraries can be selected/configured according to the specific needs of a specific application. Examples of such specific needs are: hard realtime; deeply embedded; humanoid robot; Sumo contest robots; . . .

2.3 Community goals

An appropriate level of inter-project discussion and synchronisation can lead to improved quality of all projects involved, to less duplication of development efforts, as well as to a higher personal satisfaction of the project members. (However, the pressure for projects to synchronise in this way has never been sufficiently high in the past.) In addition, the concerted, worldwide effort to create CRAMOS should result in a self-sustainable “portal” organisation (<http://www.CRAMOS.org> for example?), much like successful examples such as [FreeDesktop.org](#), the Linux Foundation (<http://www.linux-foundation.org/>), GNOME (<http://www.gnome.org>), KDE (<http://www.kde.org>), the Consumer Electronics Linux Forum (CELF, <http://www.celinuxforum.org/>), Python (<http://www.python.org>), Apache (<http://www.apache.org>), OpenOffice.org (<http://www.OpenOffice.org>), and many others.

If well managed and with intensified community and academia promotion, this could become *the* portal for (vendor-independent) robotics, which could generate sufficient revenue to fund the *CRAMOS Foundation*. Sources of inspiration for such community portals are: *Freshmeat* (<http://www.freshmeat.net>), *Linux Weekly News* (<http://www.lwn.net>), etc.

3 Strategy

To influence the development of a large software ecosystem such as CRAMOS towards a synchronised integration of many independently developed FOSS projects requires a different strategy compared to the traditional “single vendor” software development. CRAMOS can only be successful if it can fulfill the following goals: (i) to create the respect of the FOSS projects by working *with* their sensitivities and not against them; (ii) to choose the right approach at each of the various CRAMOS levels; and (iii) to build upon successful large-scale FOSS projects in more general domains than robotics and automation, even if this requires throwing away years of development in one or more of the small FOSS projects.

3.1 Social aspects of FOSS projects

It is important to be aware of the usual sensitivities in FOSS developer communities:

- many FOSS projects have their own “development roadmap” and are very hard to persuade to deviate from it, or to see the need or added value of interoperability with other software.
- many FOSS projects have spent very little time in designing their project with the large-scale application and interoperability in mind that is required for CRAMOS.
- (hence) few FOSS projects focus on “outsourcing” as much functionality as possible to more or less independent libraries, in order to increase modularity, decoupling, and (hence) reuse of the code. As a result, the APIs of most FOSS projects are too application-centric and mix too many levels of abstraction. This hinders the re-use of libraries over different applications, and leads to (poor) re-invention of the same wheels in different projects.
- many FOSS developers find it obvious that all users are capable and willing to configure and even to extend the software they want to use.
- many FOSS developers will not be convinced by “management arguments” but only by concrete code suggestions, preferably in the form of patches to their *own* software tree.
- many FOSS projects neglect documentation writing, at least to the level of ontological detail and completeness required by CRAMOS.
- many FOSS developers love to attend workshops worldwide to discuss their designs and their code, and to “hack” together during a couple of days. Free travel, hosting and food can be very motivating factors.

So, if CRAMOS would be managed by an independent organisation, this organisation’s management should include some people that know how FOSS projects work, from the inside. It would also be a good idea to let this organisation start within one or more university research groups with large critical mass in the domain of their project; this would increase the amount of good ideas and discussions that can flow into the project. Practically, speaking, this could mean to sponsor some of the FOSS projects that came out of university groups.

If an existing FOSS project cannot be convinced to make itself interoperable with the CRAMOS idea, it is better to “fork” the project than to keep on trying.

It absolutely pays off to hire only the best people for a given project, even if they are physically isolated and/or located at remote ends of the planet.

Besides the FOSS community, the CRAMOS initiative should also spend a lot of effort to convince large “end users” or “closed-source” development labs (e.g., the large Asian industrial consortia developing humanoid or service robots) to cooperate with CRAMOS, instead of (i) continuing to do everything on their own, or (ii) putting all their eggs in the Microsoft basket.

3.2 Multi-level approach

Full CRAMOS systems will contain a formidable amount of software processes, running on various CPUs connected by different networks. It is obvious that such systems will consist of very heterogeneous software parts, and that it is rather stupid to try and design a huge “one size fits all” software system. On the contrary, the key design contribution of the CRAMOS initiative will be the appropriate choices of sub-ecosystems, on, at least, the following levels:

- *Hard realtime control.* To create reliable control of automatic devices, hard realtime constraints must be satisfied at the *device level* and the *machine level* (Sec. 1); most of the control takes place at *continuous* physical and time scales; decision making is limited to realtime *state machines*, with very hard-wired *work flow*. Appropriate languages will be C, C++ and ADA.
- *Closely coupled multi-agent control.* Multiple machines, sensors, planners, etc. are to be controlled individually, but they also have to satisfy synchronisation and data exchange constraints that can become quite complex, and that are time-varying in many realworld situations. Most of the control takes place at *discrete* (symbolic, logical) scales; “as fast as possible” is the desired time latency; and the quality and flexibility of the *work flow* becomes the main design goal. The scope of these “agents” are the *machine level* and the *shop floor level* (Sec. 1). Appropriate languages will be C++ and Java, and *StateCharts* or *Petri nets* for the work flow.
- *Loosely coupled multi-agent control.* The *shop floor level*, *plant level* and *corporate level* (Sec. 1) perform control at high (symbolic) knowledge levels, with very reduced information streams, large “latencies” and complex, continuously adapted *work flows*. Appropriate languages will be Java and XML-based “business logic” work flow specification languages.

These different, complementary levels require appropriate “interconnection glue”, which will be offered by different FOSS projects.

3.3 FOSS projects to build upon

Here is a list² of large, mature and very active FOSS projects that are optimal candidates for CRAMOS to contribute to in order to reach its own goals. In fact, it’s the author’s belief that there is *only one way* to realise CRAMOS, and that is by integrating the FOSS robotics projects into these projects, because that’s where the real critical mass resides, and the robotics and automation community will *never* succeed in creating viable alternatives.

1. The *Linux kernel*, not in the least because of its support for multiple hardware platforms, hard realtime, data acquisition, and embedded systems.
2. The *Java* ecosystem, because it’s the only vendor-independent large-scale community which has a practically infinite number of developers.

OSGi (<http://en.wikipedia.org/wiki/Osgi>, <http://www.osgi.org/>) is—within that Java ecosystem—a major evolution for CRAMOS: it is a *component-based, service-oriented and standardised framework*, with multiple FOSS implementations, such as *Apache Felix* (<http://felix.apache.org>) and *Knopflerfish* (<http://www.knopflerfish.org>). OSGi provides very good support for large-scale and long-term deployment of loosely coupled software systems, with a huge amount of useful services (logging, authorisation, security, etc.). It can become the (non-realtime) *glue* to connect the large variety of *services* offered by the various robotics and automation FOSS projects. The disadvantage of supporting Java as its only native language is being worked on (via the so-called *Universal OSGi*) by CRAMOS-relevant communities such as automotive.

3. The *Eclipse IDE* (Integrated Development Environment) is a key example of the power of the Java ecosystem. Technically, it profits from an OSGi foundation to offer a very flexible, vendor-independent, and rapidly growing programming tool, without competition in the FOSS domain. Socially, it profits from the adaption by *all* Microsoft-independent industrial players in the domain of embedded and realtime systems.

Eclipse’s features go already way beyond traditional programming support: construction of graphical user interfaces, code generation, monitoring tools (such as *TimeDoctor*, <http://www.sourceforge.net/projects/timedoctor>), etc.

4. *Open standards* for engineering and computing systems, such as *UML*, *SysML*, and *MARTE*.

Eclipse is increasingly being used as a vendor-independent platform to build tools that support these standards, e.g., *OpenEmbeDD* (<http://www.openembedd.org>).

²Please contact the author if you know other relevant examples.

5. *Blender* for 3D visualisation and simulation, and maybe also for programming via its *game engine*. Blender is very mature in its 3D capabilities and it offers Python as the platform to extend the program, including the interfacing with external programs. However, for engineering purposes, some extra work is required on the Blender program, see <http://people.mech.kuleuven.be/~bruninc/blender/roadmap.html>.
6. *Maemo* for *embedded* HMI programming.
7. *HDF5* as binary data format standard. (Especially after the foreseen merger with *netCDF*.)

4 Roadmap

This Section gives a concrete suggestion about a roadmap to realize the CRAMOS goals.

1. Pre-CRAMOS phase

- to identify the existing FOSS projects that have *acquired enough critical mass* to be sustainable without the original developers. In practice, there is only one such FOSS project: Player/Stage (<http://playerstage.sourceforge.net/>). Some possible “runner ups” are: Orca (<http://orca-robotics.sourceforge.net/>), Orocos (<http://www.orocos.org/>).
- to summarize existing *project modularity and complementarity*, including suggestions for *standardization* of (sub)APIs and for sub-projects to be spun-off as viable stand-alone projects of large common interest.
- to identify *missing modules* in these projects, including the definition of the abstract APIs for these modules.
- to create one *robotics software portal* website, <http://www.CRAMOS.org>, for example, or the existing RoSta website (<http://www.robot-standards.org>). All projects in the targeted ecosystem would be linked to the portal as (independent) sub-projects, sharing news via **RSS**, sharing ontology/glossary definitions, etc.
- to summarize the *license compatibility issues*. For example, there might be a commercial reluctance to accept Orca because of the GPL licence of its ICE foundations.

This first phase is best realised by a rather small group of experts, say 5–10, in a number of intensive email and face-to-face meetings. These experts should partly come from the major FOSS projects, but preferably also from high-end and large-scale *end users*.

The outcome of this first phase is an extended version of this document with the *vision* about what CRAMOS will become, *how* its goals can be achieved, and what community support exists.

2. CRAMOS start-up phase. In this phase, the seeds for the CRAMOS ecosystem are planted, and active community and company support is stimulated worldwide.

- to create standard interfaces for inter and intra project modularity.
- to discuss common infrastructure for hardware abstraction, data flow, event handling, decision making.
- to start the FOSS projects that will fill in the missing modules in the overall design vision. These new projects should try hard to target more than just a robotics public, in order (i) to increase their critical mass, and (ii) to improve their appropriate decoupling into modules.
The human-machine interface side is a very much underdeveloped aspect of CRAMOS, while it has the largest potential to attract new people.
- to set up an *industrial interface* board, that pro-actively works together with commercial/industrial partners, to care for their needs and their problems.
- to organize a series of workshops worldwide, to connect to the academic, industrial, and service communities. These workshops are preferably linked to the “natural” big events in the various targeted communities, such as conferences and trade shows. These workshops could also have the form of summer schools or “hackatlons” (<http://en.wikipedia.org/wiki/hackatlon>), in order to attract good junior people.
- to select and configure a common build system, packaging, and a benchmarking and testing framework.

This phase of CRAMOS should be “managed” from the CRAMOS Foundation, but most actual development and refactoring should be done in the existing FOSS projects, or in newly created projects that fill identified gaps. The CRAMOS Foundation could also try to find external, targeted sponsoring, for example in the form of *Google Summer of Code* projects.

3. CRAMOS maturity phase.

- the CRAMOS Foundation should have become self sustainable.
- It should organise one or two conferences a year, in different parts of the world, and with both “developers” and “end users” focus themes.
- the first initial commercial products on top of CRAMOS should be demoed.

5 Potential cooperations from existing “meta” projects

- **RoSta** (<http://www.robot-standards.org>) which has funding to develop standardized programming interfaces, ontologies, benchmarks, and architecture templates.

RoSta has some paid people for about two years, but they lack concrete information and guidance to be optimally useful in the CRAMOS context.

RoSta already has started an overview of robotics middleware projects mentioned above, but this initiative requires more work and external inputs.

- **OpenEmbeDD** (<http://openembedd.org>) is an [Eclipse](#)-based [Model Driven Engineering](#) platform dedicated to embedded and realtime systems, working towards support for the standards [UML](#), [SysML](#), and [MARTE](#).
- **OSADL** (<http://www.osadl.org>), especially since its recent (September 2007) opening to academic partners worldwide. OSADL has focused very strictly on only the hard realtime aspects of the Linux kernel, and should now be persuaded to attract companies and academic partners that can contribute to the other CRAMOS levels.
- **OSGi** has established a number of Expert Groups to design missing parts. Two important such missing parts for CRAMOS are: (i) interoperability with other languages than Java, and (ii) interfacing with real hardware. The *Vehicle Expert Group* (VEG <http://www2.osgi.org/VEG/HomePage>) has been established to reach these goals.
- **Simtk** (<https://simtk.org>) is an ambitious FOSS project on the physics-based simulation of biological structures. It has *a lot* of infrastructure needs in common with CRAMOS.

6 Conclusions

There are no technical obstructions to create an all-encompassing “operating system” for robotics, automation and manufacturing, since all technical aspects are already covered to some extent in existing FOSS projects. The largest hurdles are social: overcoming the *not invented here syndrome* in the existing FOSS projects to reuse code from other projects, to co-develop common infrastructure, and to integrate with a (few) large FOSS projects from more general-purpose domains. The most urgent concrete actions are: (i) inter-project discussions about increased modularity of the code, (ii) definition of standard interfaces for all these modules, (iii) far reaching adoption of modelling standards such as UML and SysML for increased design (and hence code) reuse, and (vi) integration with the Eclipse/OSGi ecosystem.