

The Use of Reuse for Designing and Manufacturing Robots

Erwin Prassler, Herman Bruyninckx, Klas Nilsson,
Azamat Shakhimardanov

White Paper

on the reuse

of hardware- and software components and architectural patterns
in the design of robot systems and applications

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 3 |
| 1.1 | Component re-usability in evolving robotic systems | 3 |
| 1.2 | The “incomplete” success story of the SICK scanner | 4 |
| 1.3 | Objective of this paper | 5 |
| 1.4 | Target audience | 5 |
| 1.5 | The coming about of this paper | 5 |
| 2 | State of the art and technical considerations | 6 |
| 2.1 | Functionality concerns | 7 |
| 2.2 | Loose coupling vs system architecture | 8 |
| 2.3 | Service categories | 9 |
| 2.4 | Life cycle phases | 10 |
| 2.5 | The role of middleware | 11 |
| 3 | Technical and economic potential of reuse of hardware and software components in robotics | 11 |
| 3.1 | Platform concepts, modularity and reuse in automotive industry | 12 |
| 3.2 | Transferable and non-transferable lessons | 13 |
| 3.2.1 | Who is robotics industry and who are its customers? | 13 |
| 3.2.2 | Basic platforms, model and product variability | 14 |
| 3.2.3 | Reuse of hardware versus software components | 14 |
| 3.3 | The dilemma of usable, non-usable, and reusable components for the design competitive service robots | 15 |

| | | |
|----------|---|-----------|
| 4 | Recommendations (to the target audience) | 16 |
| 4.1 | Recommendations to Robotics Research and Education | 16 |
| 4.2 | Recommendations to Robotics Industry | 17 |
| 4.3 | Recommendations to Research Politics and Funding Agencies | 17 |

Abstract

The concepts of *reusability* and *evolvability* of hardware and software components become more and more relevant in the development of modern and future robotic systems. These much-neglected *non-functional constraints* are key enablers of the innovative, multi-vendor, high added value *robotics ecosystem* of academic research, industrial manufacturing and professional services:

- The economic value of service robotic *systems* is (among other things) proportional to the number of applications that they can serve with the same set of hardware and software components.
- In a world in which it is not yet clear what the market demands, and how these demands will change quickly, because of the relative immaturity of the service robotics field, a major factor to the economic success of service robots is their ability to adapt to changes in the requirements of the delivered services.
- This holds for, both the robotics systems *providers* and their customers who *exploit* the services' economic value.
- In order to optimize economic value, the system providers must be able to make the most optimal *system design trade-offs* for each of their *particular* service requests, when they put their end-user system together from reusable, reconfigurable *components*.
- Hence, this design optimization flexibility requirement is passed on, via the providers of *OEM components* ("marketable subsystems"), all the way down to the academic research community.

This research community currently still lacks the insights in how to realize such *stable subsystems* that are easy to reuse in robotics systems with *evolving* system-level requirements. Hence, a breakthrough is needed in the methodology and the technology that can provide simple subsystems with *the least amount of hard constraints* in their individual designs, such as to *maximize configurability* (and hence reusability) of the system-level constraints during composition, or even during operation.

1 Introduction

This Section discusses the concept of *component reusability* for *evolving robotic systems*, and introduces its opportunities and problems by means of one of its (apparently) most successful examples, the SICK laser range scanner. It uses the lessons learned from this example to give an overview and a motivation for this White Paper.

1.1 Component re-usability in evolving robotic systems

Most man-made devices of the past and the present, including industrial and service robots, are developed as rather static systems. They hardly change their functional and interaction capabilities after they leave the factory. In the context of growing awareness for and interest in service robotics, such static robotic systems will have little appeal to future manufacturers and customers, since the know-how, hardware and software components that can be used in service robots, is in full evolution. So, both manufacturers and customers will want to have the opportunity to update their (expensive!) systems with the latest features and functionalities, even when these systems are already operational at the customers' site.

In addition, the service robotics market is expected to create a very varied ecosystem of large and small manufacturers, component providers, and service operators, and most service

robots will (have to) consist of components coming from many different, competing *OEM providers*. Such a market poses new challenges on the system development process: the OEM providers, the robot manufacturers, the service providers, *and* the customers will all provide, add, change, customize, . . . parts of the whole system. So, the robotics community at large requires a revolutionary change in how it builds systems from components, starting with much tougher design, development and deployment requirements at the component level. That means that future service robotics systems engineers will have to acquire much better skills in *systems-level architecture* and *design of loosely coupled components*.

1.2 The “incomplete” success story of the SICK scanner

The SICK laser range scanner was originally developed as an *Original Equipment Manufacturer* (OEM) component for *safety systems*. It uses a scanning laser beam to detect the distance to the closest obstacle in a scan of two beams per 1 degree angle, and with a refresh rate of about 20Hz. These specifications happened to fit also well on the robotics application of detecting normal-sized obstacles in man-made environments from car-like mobile platforms that provide significant power autonomy and that can carry a fair amount of sensing and computing hardware. As such, it became the key sensor technology for a quantum leap in mobile robotics research and created an entire new market for the device.

The SICK laser scanner is an example of a hardware/software component that can be used as a **stable subsystem**. It is in itself a rather complex system with many hardware and software components, but it presents itself to the user with a simple set of mechanical, electrical and software interfaces. In addition, these interfaces remain unchanged during the whole lifetime of the system.

The success of the SICK scanner was achieved without any changes to the product’s design and without giving access to the (re)configuration of any of its components. These just happened to fit! So, while the SICK scanner is a very specific but illustrative example for the economic and technological impact of the reuse of OEM components, this reuse was never intended by the manufacturer. It was a “lucky shot” that the specifications of one application domain matched that of another, new application domain quite well.

The SICK scanner was never **designed** to be a reusable part of an *evolving* robotic systems, since it has never been **open for modification, adaptation or reconfiguration of its internal components**. This has prevented many other useful applications, such as:

- selective scanning, i.e. scanning only those regions the application wants to focus on, with the appropriate scan frequency, accuracy and laser beam power.
- the ability to be built in other physical form factors, and with other electronic communication protocols.
- various on-board signal preprocessing on the available computing hardware.
- *peer-to-peer* communication mode instead of the *master-slave* mode that is the only one offered now. In other words, the SICK scanner *imposes* its data and its communication policy on its users.
- scanning in the third direction, not by tilting of the whole device by an external actuation, as is necessary now, but by tilting only the much lighter beam unit inside the device.

There is a plenitude of hardware and software components (cameras, motors, etc.), that have a similar scientific and economic potential which is not used because the robotics community at large (universities, small and large manufacturers, robotic service providers) is **not familiar with the concept of re-using** hardware- and software components, or architectural patterns in robotic systems design. Even more so the concept of designing these components in such a way that they have **loosely coupled** design requirement implementations, and hence that they are highly re-usable.

1.3 Objective of this paper

This work wants to create awareness for both the problem *and* the opportunities of re-usability of components in evolving robotic systems. It identifies the technical issues involved, as well as the scientific and economic impact and benefit of reuse. Finally, it provides a set of measures, practical guidelines, and desired activities, to establish and to promote the reuse of components in (i) robotic *systems* design, and (ii) the design of reusable *OEM components* in and for robotics.

1.4 Target audience

This paper wants to target primarily the following audience:

- robotics research: reuse reduces effort for secondary R&D (to build and to maintain research platforms), allows rapid development of new platforms, and results naturally in better decoupled designs;
- robotics *product* industry: reuse of hardware and software components shortens *system* development times, reduces *system* development costs, and increases the *system* qualities of robustness and evolvability;
- *OEM components* industry: make them aware of the fact that their components could be technology enabling new robotics products and markets (e.g. Roomba, service robotics); reusable components create larger market potential and allow for small-scale component *implementation* specialization;
- politics/funding agencies: support in developing and establishing concepts for reuse in robotics will lead to a boost in technology transfer and innovation; this White Paper presents some concrete fundamental and applied research topics in this context.

1.5 The coming about of this paper

Under the guidance and with the support of the European Coordinated Action RoSta (Robot Standards and Reference Architectures) an international study was established in 2007 to investigate the measurability and comparability of robot architectures and middleware.

The work of this study group lead to a number of interesting insights:

- The robotics community (research as well as industry) recognizes the importance of measurability and comparability also in the area of robot architectures and middleware.
- The community, however, strongly opposes any effort to establish a standardized reference architecture in spite of similarity and commonality of structural patterns and

design principles across many proposed architectures (Bill Smart: “*I will never use your architecture!!*”)

- Experts/developers fear to lose flexibility and variability in the design of the overall system necessary to account for requirements imposed by application and/or user; no reduction of design space
- There was broad agreement in the expert group, however, that on a component level (hardware as well as software) harmonization of components and their interfaces is most desirable to enable measurability, comparability, and above all re-usability of components.
- This would allow to speed up development process significantly still allow design variability on the system level also limit the effort for development of research platforms.

2 State of the art and technical considerations

Designing a “system” (be it a “sub-system” component, or a complete system) involves making **technical and economic trade-offs** between a multitude of design specifications: cost, energy consumption, communication and computation overhead, performance, size, robustness, etc. Secondly, the system is given an **interface** for other systems or people to use the functionalities offered by the system.

The result of the above-mentioned trade-off is determined—to a large extent even, in the context of robotic systems!—by the knowledge about the *environments* the system will be used in, and the *tasks* it will have to perform in those environments. (A task imposes *artificial constraints* on a system, whereas the environment and the robot provide real constraints.) In practice, designers typically have only their own rather limited set of environments and tasks in mind when making the trade-offs. But, more importantly, they are hence tempted unconsciously to make too many commitments in *fixing* their trade-offs and their interfaces into the design, although there is no hard physical or functional need to do so. This results in **too tightly coupled** component and system designs: it is close to impossible to undo the design trade-offs that have been made in the component, and to reconfigure them to achieve another trade-off that is more appropriate in a new task or environment context, or in a new system. As a result, most designs come on the market with much less flexibility than possible, which compromises later reuse in contexts where the “best” trade-offs lie somewhere else than what the original designers had in mind.

One also has to distinguish on what level trade-offs are made. In software market one can differentiate among three products with different requirements in mind.

- **Custom software** - is, as described above, a fairly common approach in robotics community. The reasons for this are quite clear, users/developers want to have complete control over their system and custom software solutions provide these flexibility and the advantage is that it could be completely tailored to the business model of company where it is developed and thus provides a competitive edge over other competitors. The severe disadvantage is that it is too expensive to develop from the scratch!^[4]

- * Advantages - can be tailored to the needs of a user

- * Disadvantages - expensive

- **Standard software** - outsourced software, the whole application software is developed by the third party
 - * Advantages - no risks of financial failure, limits development, integration and market time (provides exact estimates what happens and when)
 - * Disadvantages - all competitors have the same software, no any advantage over other. May require re-structuring business process to gain competitive edge or fit to the needs of the standard software [4].
- **Component software** - only parts/sub-systems of the software are outsourced
 - * Advantages - allows advantages from previous two with some trade-offs (modifications/changes can be carried out on sub-system level). It is standard software on the level of component (model, interfaces etc) but it can be customized through component configuration and integration/assembly process.

This paper structures its discussion (about the re-usability of designs, components, and systems in the context of evolving robotic systems and services), around a set of largely orthogonal design aspects, that are introduced with a *system-level* point of view in the following subsections. The presented set of structured concepts is expected to give the readers of this white paper enough insight in the *design for re-usability* problem, in order to make motivated decisions in their future component developments, purchases or funding. This approach can be summarized in two major design guidelines: **loose coupling** (sometimes also called “separation of concerns”) and **separation of component functionality from system architecture**.

The presented conceptual structures are *orthogonal*, in the sense that system designers have, ideally, to take them all into consideration at the same time. However, this document can only present them in a sequential order, so the reader might have to iterate back and forth over the following subsections before being able to grasp the whole picture.

2.1 Functionality concerns

In general, each “system”—as well as each component of more than the most trivial complexity—has functionalities that fall within one of the following four categories [2], each requiring different design-time concerns:

- **Computation:** this is the core of the component, that provides an implementation (in hardware and/or software) of knowledge that adds value to the component. This knowledge processing activity typically requires “read” and “write” access to *data* from sources outside of this component, as well as some form of synchronization between the computational activities in multiple components.
- **Communication:** this functionality is responsible for bringing data towards computational activities with the right *quality of service*, i.e. time, bandwidth, latency, accuracy, priority, etc.
- **Configuration:** this functionality allows users of the computation and communication functionalities to influence the latter’s behaviour and performance, by setting properties, determining communication channels, providing hardware and software resources and taking care of their appropriate allocation, etc.

- **Coordination:** this functionality determines the system-level behaviour of all cooperating components in the system (the *models of computation*), from appropriate selection of the individual components' computation and communication behaviour.

One often-encountered term in modern (computer) systems engineering is *orchestration*; in the terminology of this document, orchestration is nothing else but the combination of *configuration* and *coordination*.

This conceptual system functionality model has the advantage of being simple, while still capturing the essential system-level properties of complex engineering systems. The important insights that developers should get from this simple model are:

- Component developers must keep all four concerns as separate as possible. For example, design and development of the Computation in a component should be done without information about how exactly the other concerns will be realised.
- The Computation components should ideally consider *synchronous* data exchange only. In other words, the writer of the computational algorithm can assume that its algorithm can read and write all the data that it needs, without having to worry where the data comes from, or whether it is “fresh”.
- Architectural trade-offs (see Section 2.2), are done *only* in Configuration and Coordination. For example, the above-mentioned *synchronicity assumption* in Computation should be realized by appropriate Configuration (of data Communication between Computational components) and Coordination (of time behaviour of the Computation and Communication components).
- Coordination can be achieved by a *fully domain-independent* set of primitives and software tools. These primitives are basically nothing more than the well-known activity and state synchronization concepts of *Finite State Machines* (i.e., single-activity behaviour coordination), and *Petri Nets* (i.e., multi-activity behaviour coordination, which boils most often down to *synchronization* of concurrent activities).

The **SICK laser scanner** is a typical representative of a subsystem that has not been designed with the above-mentioned concerns in mind: it offers only one possible Communication mode with limited Configuration options, it has only one Computational functionality (at least, as far as visible by an independent user), and no Coordination possibilities (it works as a constant server of data, and it's the client's responsibility to keep up with that data stream).

2.2 Loose coupling vs system architecture

By definition components are products of **third party composition** That is, it should be decoupled of concerns that it is not supposed to take care of. The other important point to make is that whenever one talks about software, it is often software as a **meta-product**, but deals with (runs on a system) its **instance**. One should not forget about this. Concerns such as configuration, coordination are closely related to particular instance of a component, whereas Computation and communication to its **plan** (as mentioned in section above, these concerns can be observed at different phases of component life cycle) [4]

Component builders should work hard to offer their components with:

- as few design requirements rigidly fixed as possible.
- a Configuration interface through which the design trade-offs can be adapted to each particular system’s context.
- *loosely coupled* functionalities, that is, the amount of “*information*” that the developer of one part of a component has to have about what is happening in other parts should be minimal. This is a *design time* requirement, which also has a corresponding *runtime* requirement: the amount of communication between the two components should be as low as possible.

In other domains than robotics, these principles are commonly known (but often not worked out to the same level of concrete detail, or with a smaller scope) under many other names, such as: *decoupling mechanism and policy* in operating systems; *Demeter’s Law* in software engineering; the *Principle of Compositionality* or *Frege’s Principle* in semantics and philosophy of language; or *stable subsystem* in holonic manufacturing systems [1, 3].

The corresponding design guideline is to subdivide (“re-factor”) component libraries into smaller parts as long as the information exchange between the parts reduces too. At a certain moment, one will notice that the inter-component information exchange tends to increase with smaller component size, because the knowledge in both components has a *natural coupling* that makes them go together in the same component.

System architects play a role that is highly complementary to the role of component builders, since **architecture is all about making coupling trade-offs**, using the configuration interfaces of **components that have been designed with loose coupling in mind**. The system architect is responsible for the (online re-)configuration of the components into a coherent system that satisfies the (evolving) system requirements, and for the coordination of the computational and communicational behaviours of those components.

The **SICK laser scanner** is a typical representative of a subsystem that is not made available as a loosely coupled set of components. (No information is available about how the company has done the internal design and development of their laser scanner.) So, it has very few configuration options, since it has rigidly fixed (the implementation of) the design specifications in all other concerns.

2.3 Service categories

In general, each “system” provides its functionalities in three complementary forms:

- **Object/class library**: this are pieces of (compiled, interpreted,...) code that are available to all functionality categories in Sect. 2.1. **Linking** is the relevant architectural, system-design activity around class libraries; this typically involves (only) Configuration functionality.
- **Component**: in addition to class libraries, components have the notion about (multiple) activities and their synchronization. For the system architects, this means that they have to consider how **to deploy** components in a particular system, in order to start and stop the Computation of the components in the appropriate way, and to provide the appropriate Communication and Coordination. Typically, the system architects have full knowledge about which (kind of) components will be present in their systems

(even if their systems are not static) and with which data and message formats the components will communicate and synchronize.

- **Service:** the added system-level complexity of services with respect to components is that the system architects do not know in advance which services will be used, and where they are located. Hence, this adds the requirement of **service discovery** to the system design problem, including the aspects of **semantic interpretation and translation** of data and message interfaces of services.

Service robotic systems naturally evolve towards the component and service models, while our engineers have not yet been exposed much to the design and development problems that come with these models.

The **SICK laser scanner** is a component with its own activity (running on the embedded control hardware) but with no means at all for the system builders to make this component adapt to the evolving requirements and contexts of their service robotics systems.

2.4 Life cycle phases

In general, each “system” goes through the following phases at least once during its life time:

- **Design:** decisions are made about which object classes, components and services are needed, what their Computation, Communication, Configuration and Coordination specifications have to be, and to what extent these specifications should be made available via decoupled interfaces.
- **Implementation & part testing:** the design decisions are transformed into working software and/or hardware parts.
- **Sub-system assembly—Link time:** some finished parts are connected into larger parts. The smaller parts can come from different developers, so some integration problems will already be discovered here.
- **Deployment:** the finished components are made ready to serve the system users, in a particular component or service configuration.
- **Runtime:** the system is busy providing its service to its users.

During these complementary phases, **very different** aspects of the system’s design and implementation are being stressed, and different (re-)Configuration and Coordination functionalities are needed. In addition, service robotic systems will spend most of their operational time in the deployment and runtime phases, the latter one being the one where most of the economic added value will come from, but also the one with the largest **dynamism**, that is, it has the highest requirements with respect to evolvability and (runtime) component reuse and reconfiguration. Hence, our engineers should get appropriate education and training to develop (and to support!) these phases.

2.5 The role of middleware

Middleware is an established concept in large-scale software systems, and, even in the relatively small domain of robotics, several middleware projects exist, with *open source* as well as proprietary licenses. The minimalistic definition of the term “middleware” restricts its scope to the Communication concerns of Sect. 2.1. A somewhat less restrictive definition (which fits well in the context of this document) puts all application-independent *infrastructure* software within the scope of the term. So, the structure and the insights presented in the previous subsections lead to considering the following functional concerns as primary candidates for middleware efforts:

- Communication. This requires no further explanation, because it is the traditional subject of middleware.
- Computation, as far as really **cross-cutting** functionalities are concerned, such as instrumentation, reporting, automatic code generation, etc.
- Coordination. This is an extremely valid candidate, since, in its ideal incarnation, it deals only with fully application-independent issues. Developing it in a separate middleware project would also reduce the chances of creating coupled designs and implementations.

Two existing examples that come already close to this description are the various implementations of the **OSGi** and **CORBA** standards.

3 Technical and economic potential of reuse of hardware and software components in robotics

While the reuse and composability of hardware and software components has received quite some attention in the academic robotic community recently, primarily in the area of robotics software engineering, these concepts do not play a very important role in the robotics industry yet.

As a matter of fact it turned out virtually impossible to get any valid economic figures, which might illustrate whether or not the reuse of components across the product lines of a single company or the use of so-called OEM components¹ plays any significant role in the development and manufacturing of robotics systems.

Not even the *International Federation of Robotics* (IFR), which monitors the development of the industrial robotics market at large and maintains comprehensive statistical material on this market, or its European counterpart *EUnited Robotics*, has any figures in this direction available.

This may be considered as a clear indication that the (re-)use of in-house components or OEM components is not considered an issue. For the “traditional” robotics industry there may be in fact limited use for reuse. As we will show, however, the reuse of components,

¹Note that in industries such as automotive or aeronautical industry an OEM (Original Equipment Manufacturer) “is typically a company that uses a component made by a second company in its own product, or sells the product of the second company under its own brand. The specific meaning of the term varies in different contexts” (http://wikipedia.org/wiki/Original_equipment_manufacturer). In the IT industry, the term OEM also denotes a manufacturer of a component or a subassembly used in the production of a larger item. In this paper we use the term common in IT industry.

especially of OEM components, which were developed for other applications and markets, can become a key factor for the design of affordable service robots.

In the subsections below, we will first look into a well and widely known prime example for the economic as well as technical potential of reuse of components and platforms. We will further discuss, if and which lessons can be learned and transferred to robotics industry. In a last subsection we will illustrate in a semi-fictitious example application the dilemma regarding the usability, non-usability, and re-usability of components and the impact which this dilemma has for the “new” service robotics industry.

3.1 Platform concepts, modularity and reuse in automotive industry

Design strategies, which support re-usability and composability of components in product design, are reasonably established in many industrial areas, starting from car industry, over embedded systems industry, to aircraft industry. Often it was the economic pressure and not so much technological insight, which finally paved the way for the strategic decision to promote re-usability and composability of components.

In 1994 Volkswagen introduced the so-called “Volkswagen platform concept” at a time, when the the competitiveness of the company, the quality of its products and consequently its earnings was steadily increasing, while the cost were steadily increasing and the company altogether experienced a severe economic crisis.

At the core of the Volkswagen platform concept was the plan to design the 20 models of the different Volkswagen brands such as Audi, Skoda, or Seat, on top of four re-usable basic platforms (one, for example for Audia A4, VW Passat, Skoda Octavia, one for VW Golf, Seat Ibiza). In the years following this decision Volkswagen could decrease its production cost significantly and got back into the profit zone. In the sequel the platform strategy was developed further, promoting the reusability of components and compound components, and also the profitability the company further increased.

In 2000, Ferdinand Piech, the CEO of Volkswagen, announced that the company would invest significant effort in the development, production of components and compound components re-usable across different models as well as different Volkswagen brands (Deutsche Welle, 7. Nov. 2000, 00:00 Uhr: “*VW spart mit Verbundsystem Milliarden.*”)

Volkswagen estimated that the savings due to this new strategy would total to two to three billion Euros per year. They expected a saving of around 500 Euro per car.

In their new strategy (“Verbundsystem”) the company planned to increase the multiple use of parts by a magnitude. Volkswagen said that with this “Verbundsystem” a heavily powered VW Polo, a medium powered VW Golf, and a moderately powered VW Passat could be built on very similar components in terms of engine, break system and axle suspension.

Volkswagen also managed to avoid a major drawback and risk, which could be a side-effect of re-using components across different models: watering the uniqueness of the brand and by doing so displacing and disappointing faithful customers who have bought the some model for years and decades. Volkswagen managed in spite of many common components to give their different models and brand unique designs.

This leads to a rather important insight and conclusion for promoting re-usability of components: unique design (architecture) and look of the final product and subsequently the uniqueness of the brand do not exclude re-usability of components at a large scale and vice versa.

In summary, one can say that the promotion of the platform concept and later the emphasis on the usability of components across several models and even brands has reduced the production cost in the order of three to five percent, without effecting the uniqueness of the brand.

In the same above-mentioned press release, Volkswagen also announced that within five years the company intended to increase the number of basic platforms from four to eleven and the number of models up to 67. Although this was not part of the official statement one can speculate that the VW modularity concept also had a noticeable impact on the variability of the Volkswagen product line.

3.2 Transferable and non-transferable lessons

In the preceding paragraphs we have described an show-case, where the introduction of a platform concept and the concept multiply (re-)usable and configurable components has had a tremendous impact on the economic situation of a very large enterprise. This strategy and variations of it have been picked up by almost all big players in automotive industry and the coverage in the media has even created something like a public awareness for platform concepts, (re-)usability and composability of components.

The obvious questions are now: Can this strategy straightforwardly be transferred to robotics and the robotics industry and what would the economic benefit be? The general answer is straightforward: Why should it not be possible to transfer the lessons and insights of the automotive industry to robotics industry and why should there not be similar economic benefits? This general answer does what general answers typically do, it abstracts from the details, which may affect the truth of the answer.

3.2.1 Who is robotics industry and who are its customers?

The term “robotics industry” is often taken synonym for “manufacturer of industrial manipulators, whose main customer is automotive industry”. Although this is a simplification and no robot manufacturer today can afford to exclusively serve automotive industry, there is one criteria, which discriminates the traditional robotics from the so-called “new” robotics industry: the first one does not manufacture bulk products, while the second more and more does.

The “new robotics industry” – the term also tends to oversimplify the situation somewhat – is formed out of enterprises which manufacture service robots and robotics appliances. Although this industry is still vanishingly small, it does serve a similar market than the automotive industry: *the mass*. Although at present the annual turnover of both industries may not differ by a magnitude, the sold quantities in fact do. For example the number of cleaning robots sold over the past five years is said to total to around 10 million units.

Relevance of re-usability and composability for non-bulk products: Although a strategy to modularize the design of industrial robots and to reuse components across different models of industrial robots is not per se irrelevant, the economic impact of such a strategy may be limited. Robot controllers are moderate examples for a reuse of components in industrial robotics. Often robot controllers are not designed from scratch but are parameterized and can be adapted to different manipulators. The same may be true for other components of an industrial robot.

Relevance of re-usability and composability for robotic bulk products: The situation is entirely different for the design and manufacturing of robotics bulk products, for example robotic appliances. If a domestic cleaning robot is sold by the millions per year then a saving a five Euros makes a difference, no matter whether this saving comes from the re-usability of a component, which was initially designed for a different mass product and therefore can be purchased at lower cost, or whether a component can be used across different products of the same company and therefore both reduce cost for production and purchasing. In Section 3.3 we will discuss this issue in more detail.

3.2.2 Basic platforms, model and product variability

A major aspect of reuse of components in automotive industry was to achieve a maximum variability in the final product family with a minimum of basic platforms and carrying reusable major components such as engines, gearboxes, break systems. In this context, re-usability and composability have tremendous cost saving effects with respect to development and production cost and also with respect to purchasing material.

Product variability and reuse in traditional industrial robotics: Although all robot manufacturers offer a remarkable range of robot manipulators, the potential of reusing, for example, the same motors or gearboxes across a number different manipulators is limited and so is the potential of significantly cutting the cost. This is due to the fact that industrial robots are investment goods with moderate lot sizes and not bulk products. Product variability in the industrial robotics market is most likely due to customization request and not so much the result of a marketing strategy to create new customer demands.

Product variability and reuse in the service robotics industry: Many service robots will be specialized or even customized products, e.g. in medical or rehabilitation robotics. Users may demand a high product variability while the market size for these products will be in the same order of magnitude as for industrial robots. So the situation concerning the reuse of components and OEM components may be comparable.

A great deal of the service robots, which one can picture today, however, are and will be bulk products. The market launch of the low-cost cleaning robot *Roomba* by iRobot Inc. in 2002 was a historic event for the commercialization of robots as consumer goods. Six years later iRobot claims that they have sold four million of their home floor cleaning robots since then (report at annual meeting of to stockholders in 2009)². These figures suggest a market size which is comparable to that of any other advance appliance.

Assuming that the total production cost of a robot like *Roomba* with a retail price between 300 and 500 EUR is in the range of 100 to 150 EUR then it becomes apparent that component cost and their reduction are rather essential issues. Particularly the (re-)use of OEM components designed for other bulk products can help to reduce cost significantly.

3.2.3 Reuse of hardware versus software components

In the preceding discussion we wanted to illustrate that reuse and composability of components can have a noticeable economic effect and lead to non-negligible cost reduction. At the same time we also wanted to point out, that reuse and composability are not panaceas, which remedy any high-cost structure or low-market volume or acceptance.

²The number of new industrial robots installed in the year 2007 worldwide was counted to 114.365.

In our discussion the focus was on the reuse of hardware components. This was not by chance. The reuse of hardware components and the reuse software components do not have exactly the same flavor. Manufacturing hardware involves production tools and facilities and supply of parts. These represent a significant cost factor. Neither the production facilities and tools nor the supply chains can be changed too frequently. Accordingly hardware products and their components do have much longer cycle times than software. Any decision towards reuse and composability of components is therefore strategic with a respective time horizon.

In software development and engineering decisions towards component reuse can be much more exploratory and flexible. At the same time reuse of software components can speed up the development process and reduce the development time for new robot applications remarkably.

3.3 The dilemma of usable, non-usable, and reusable components for the design competitive service robots

More than four million domestic cleaning robots sold over a period of five years is a clear indication that cleaning robots are no longer considered as exotic and futuristic devices but as a new generation of appliances. A major reason for the success of these devices is of course their low-cost (< 500 EUR). Apparently the regular consumer is willing to invest a certain amount of money in new products, even if their ultimate use is not entirely known.

If it comes to professional cleaning, the situation is rather different. Professional cleaning leaves no room for gadgets. The expectations of the targeted customers – facility managers, cleaning service providers, municipal administration – are rather mundane. They expect professional cleaning service. This in the first place means systematic coverage and quality of service at competitive cost, accounting for constraints imposed by the management of the facility.

The technical requirements which this entails for a professional cleaning robot are

- robust 3D obstacle avoidance in partially unknown environments
- accurate localization over large distances and areas
- systematic coverage of the work (= cleaning) space
- safe operation

The market for indoor cleaning machines in 2009 has been estimated to 1,145 billion Euros. The market for professional cleaning service in total only in Europe is estimated to several ten billion Euros per year.

In the face of such figures it has been extremely tempting and challenging to automate at least a share of this market by means of professional cleaning robots. Unfortunately automate professional cleaning by means of robots looks back to a long series of failures. In the past 20 years we have seen more than a dozen attempts even by large enterprises to develop professional cleaning robots. They have all failed miserably.

The reason for that is not that the above technical requirements are beyond what robotics research and development can provide today as state of the art. The reason for that is that most of these professional cleaning robots had a (tentative) sales price of around 50.000 EUR. In contrast a non-robotized, human operated comparable cleaning machine costs in the order of 10.000 to 15.000 EUR. Apparently the cost for robotization dominated the sales price and led to (prototypes of) products which were far from being competitive. As a rule of thumb

one can say that the additional cost for the robotization of a cleaning machine must not exceed 15% of the final cost. This means that robotizing a cleaning machine which is in the range of 10.000 EUR must not cost more than 1500 EUR.

Now, if we look into previous designs of cleaning robot then we will find that only the cost for the hardware components used today for environmental sensing and for accurate localization exceeds this amount already by a factor of three or four. Apparently there is a rather fundamental dilemma.

Unfortunately, this dilemma is not only limited to professional cleaning robotics. It actually holds true for large parts of service robotics: the established hardware components used to implement basic robotic functionalities such as robust 3D obstacle avoidance or reasonably accurate robot positioning over large distances and areas exceed by far the price range which would make the respective service robot a competitive product and reasonable investment.

A potential answer to this dilemma is a systematic exploration of components from other industries which manufactures bulk products such as gaming industry, automotive industry, multi-media industry for their use and reuse for robotics products. These industries have already in the past developed components which have been used as OEM components in robotics and there seems to be a tremendous potential for many more low-cost components and sensors, that may in fact enable many new service robotics applications.

Such a systematic exploration can, of course, be left to individual actors, since they will be the beneficiaries in the end. At the same time the above example shows that although the technology from a scientific point of view is available to design versatile service robots, this technology has no real use since in its current form is it has no enabling effect. It is way too expensive to find its way into marketable product.

Therefore a joint initiative between political and administrative decision makers, industry and the academic robotics community is necessary, to escape from the described dilemma.

4 Recommendations (to the target audience)

This Section provides a short but focused list of recommendations for actions to be taken by the various stakeholders in the target audience of this document.

4.1 Recommendations to Robotics Research and Education

- Academic engineers must receive improved teaching and training in system and component level design and architecture.
- More research is needed to identify and develop appropriate computer support for system level design and architecture, as well as for runtime deployment, configuration and service discovery.
- The academic robotics community should (preferably in cooperation with industry) identify the “best practices” in the domain of the APIs for, at least, (asynchronous and synchronous) method calls between components, asynchronous Events, and (synchronous and asynchronous) data flows.

These best practices should end up in software support toolchains, and in vendor-independent international standards.

- The academic research community should come up with a wide range of quantitative measures to assess all relevant component and system design trade-offs, including their resulting runtime execution performance.

4.2 Recommendations to Robotics Industry

- The robotics industry should start standardizing some component interfaces, but start from the most mature ones at the “bottom” of the software and hardware infrastructure.
- The robotics industry should bring order into the abundance of communication middleware, including field buses.
- There should be technology transfer stimuli to foster the creation of small-scale, reusable components, by independent vendors, because that’s where most of the new added economic value will come from.
- This requires that the traditional robotic industry makes a serious effort in opening their systems and development processes.

4.3 Recommendations to Research Politics and Funding Agencies

- support more careful and comprehensive market studies
- support compilation of general guidelines for development of re-usable components
- impose on European robotic research projects to develop a strategy on reuse of components.

References

- [1] Koestler, Arthur. *The Ghost in the Machine*. 1967 (reprinted Penguin, 1990).
- [2] Radestock, Matthias and Eisenbach, Susan. *Coordination in evolving systems*. Trends in Distributed Systems CORBA and Beyond, pp. 162–176, Springer, 1996.
- [3] Simon, Herbert. *The Sciences of the Artificial*. MIT Press, 1969.
- [4] Len Bass, Paul Clements and Rick Kazman *Software Architecture in Practice*. Pearson Education Inc, 2007