

# A Telepresence Robot System Realized by Embedded Object Concept

Tero Vallius, Juha Rönning  
University of Oulu, Department of Information and Electrical Engineering  
Erkki Koiso-Kanttilankatu 3, 90014 Oulu, Finland

## ABSTRACT

This paper presents the Embedded Object Concept (EOC) and a telepresence robot system which is a test case for the EOC. The EOC utilizes common object-oriented methods used in software by applying them to combined Lego-like software-hardware entities. These entities represent objects in object-oriented design methods, and they are the building blocks of embedded systems. The goal of the EOC is to make the designing embedded systems faster and easier. This concept enables people without comprehensive knowledge in electronics design to create new embedded systems, and for experts it shortens the design time of new embedded systems.

We present the current status of a telepresence robot created with second-generation Atomi-objects, which is the name for our implementation of the embedded objects. The telepresence robot is a relatively complex test case for the EOC. The robot has been constructed using incremental device development, which is made possible by the architecture of the EOC. The robot contains video and audio exchange capability and a controlling system for driving with two wheels. The robot is built in two versions, the first consisting of a PC device and Atomi-objects, and the second consisting of only Atomi-objects. The robot is currently incomplete, but most of it has been successfully tested.

**Keywords:** Atomi-objects, Embedded Object Concept, object-oriented, embedded system, Telepresence robot

## 1. INTRODUCTION

This paper presents the Embedded Object Concept (EOC) and a telepresence robot system, which is a test case for the EOC. The following chapters will shortly present the motivation and related work regarding the subject of this paper.

### 1.1. Motivation

Traditionally, embedded system design development consists of the following phases [1]:

- requirement specification
- partitioning of the design into its software and hardware components
- iteration and refinement of the partitioning
- independent hardware and software design tasks
- integration of the hardware and software components
- product testing and release
- on-going maintenance and upgrading

The problem with this process is the need for expertise and time. The system is usually built from individual components, which provides a huge number of different possibilities for implementation, and thus enables the creation of tailored solutions for the systems at hand. At the same time, it requires a lot of time and expertise on embedded system design to find suitable components. Trial and error or incremental development are not feasible approaches in electronics, since the hardware is very expensive to modify once it has been made. Usually, a robust simulation and prototyping is needed in the partitioning phases to confirm the partitioning and component selection decisions. Still, there is room for errors as the prototyping hardware is often just an emulation of the actual implementation hardware, and the electrical characteristics of the design are often based on experience with previous projects.

The goal of our research is to make designing embedded systems easier and faster. For this goal we have proposed the Embedded Object Concept (EOC), which approaches this problem via object-oriented methods and physical electronic modules called embedded objects [2]. Embedded objects are tiny printed circuit boards (PCBs) containing both the hardware and software of an object, thus being complete functional entities. These objects can be used as building blocks of an embedded system. The objects can be interconnected to one another like Legos. Each object has a

generalized and well-defined interface. For building new devices with the embedded objects, suitable objects are connected together and some control code is written. For a more complex design one can apply object-oriented design principles. The EOC enables fast prototyping and easy modifications, and it enables incremental development of devices, which is impossible with traditional design methods.

This paper presents the current status of the EOC and the telepresence robot test case. We have implemented two generations of embedded objects. Our implementation of the embedded objects is named Atomi-objects. The telepresence robot is implemented with the second generation of embedded objects. The telepresence robot test case aims to validate some EOC-related issues: the suitability of the EOC for relatively complex systems, the functionality of the Atomi II –framework [3], object-oriented design, and incremental device development.

## 1.2. Related work

Different object-oriented embedded system approaches exist. The object-oriented methods have been applied to embedded system co-design in MOOSE (Model-based Object-Oriented System Engineering) [4]. Object-Oriented principles have been used for hardware design and hardware synthetization in SystemC [5] and object-oriented extensions to VHDL [6]. Some more concrete approaches are OOPic, RoboBricks, SimmSticks and the Tower system. OOPics are PIC-microcontrollers with preprogrammed multitasking objects from a library of software objects, and they are mainly meant for robotic applications [7]. RoboBricks are quite similar to OOPic, but they are complete modules consisting of electronics and software. RoboBricks create master-slave types of architecture and they are also meant for robotic applications [8]. SimmSticks are electronic modules that can be inserted into a motherboard to create computer types of embedded systems. They are aimed for hobbyists building quick custom devices. The SimmSticks do not contain any preprogrammed software [9]. The Tower system consists of modules that include a foundation board and a set of layers to provide the required functions. It is very similar to RoboBricks. The Tower system is meant for modeling a system topology [10, 11].

## 2. EMBEDDED OBJECT CONCEPT

The basic idea of the EOC is to build new devices in a manner similar to building new toys with Legos. A set of ready-made building blocks can be connected to each other to create a new device. In EOC the building blocks are small electronic boards, called embedded objects, which interconnect with each other via a general purpose bus. Each board has a different functionality, such as servo control or sensor inputs. A new embedded system prototype is built by connecting together boards that have suitable functionalities for the new system, and then creating high-level control software to give the system the intelligence. This process is depicted in Figure 1. It also shows images of our implementation of the embedded objects, the *Atomi*-objects.

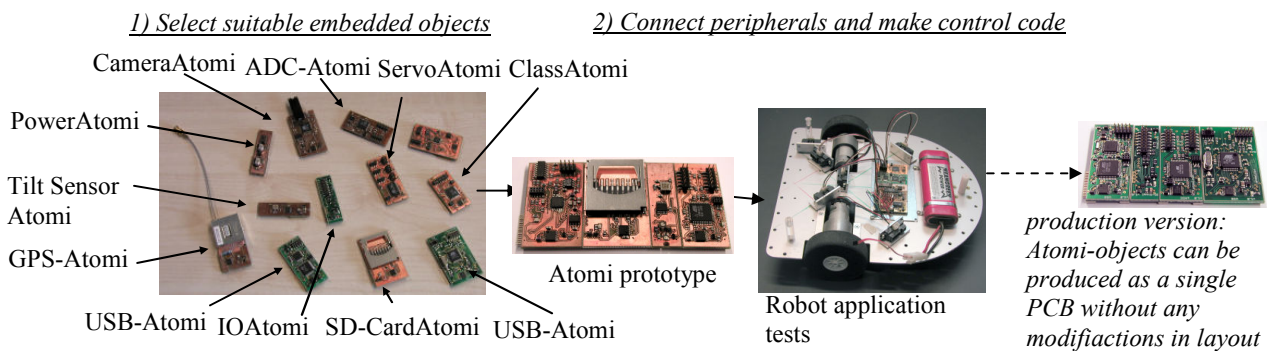


Figure 1 Using embedded objects

The embedded object concept applies object-oriented methods into embedded systems. The novelty of this concept is that we consider the objects to be these physical Lego-like electronic boards. Object-oriented fundamental elements, such as modularity, encapsulation, abstraction, hierarchy, inheritance, and concurrency can be applied to these objects [2]. By utilizing object-oriented principles the EOC has the well-known benefits of object-oriented systems, such as maintainability, reusability, stability, reliability, faster designing, and extensibility. Furthermore, the embedded objects

enable us to use ready-made building blocks for creating new systems, which contributes to the easy-to-use aspect by removing in many cases the need, in many cases, for creating new electronics and low level software at all.

### 3. EMBEDDED OBJECT ARCHITECTURE

The EOC is based on an inter-object bus and an object-oriented logical interface between objects. The EOC generates object-oriented architectures. Thus, common object-oriented design methods can be used with complex systems. The design can be notated by UML or other object-oriented notation languages.

#### 3.1. General architecture

The high level architecture is very simple. A device made with EOA consists of objects. These objects can be constructed from a group of other objects according to object-oriented principles. The simplest object is a single physical entity, which performs only one function. This is the basic principle for partitioning new functions into embedded objects.

A basic embedded object consists of a small PCB with at least one bus interface and some functional parts, for example an AD-converter or motor driver. Each object can be connected to any other object. Some objects have two or more buses, and they can be used to separate the buses between different object groups and create new interfaces for them. These objects are called class objects, as they represent a new class consisting of the objects connected to them. By dividing the objects to separate buses, the system can be divided into new logical objects that consist of a group of other objects, which is the basic principle of object-oriented complexity growing [12]. Figure 2 shows an example of an architectural diagram of such a system.

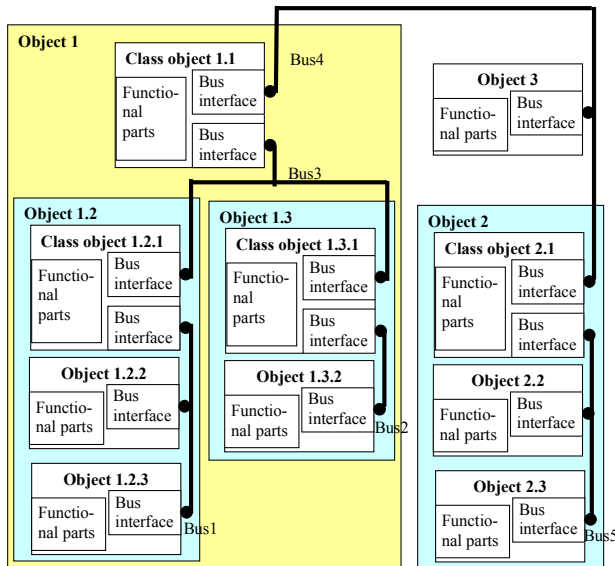


Figure 2 Architectural example of the EOA

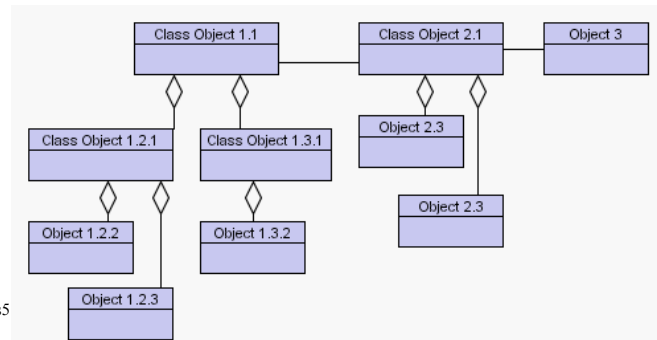


Figure 3 Example of UML notation

The EOA is basically a networked multiprocessor system that is implemented in low level electronics. The bus interface of each object corresponds to the logical interfaces of the objects in object-oriented design methods. Figure 3 shows the architectural diagram of the architecture in Figure 2 using UML notation. This is the basis for object-oriented design methods for EOC. To create a new embedded system, one can make the design with UML and map it directly to embedded objects.

#### 3.2. Logical interface

To enable ease of use of any object, there exists a generalized interface that is implemented into the default software of each object. The communication between objects is based on the idea that each object can have properties, methods and events. Properties, method triggers and event setups are considered to be public variables of an object according to

object-oriented principles. These variables are shared through the bus interface. Each variable represents some item of the object, for example the value of an IO port or the speed of a DC motor. Objects are addressed by their object number, which identifies the object. In more electrical terms, each object has a bus address (i.e. object number) and a set of registers (i.e. variables).

TABLE 1 shows an example allocation of variables in a User Interface Atomi. It has properties for the text of the LCD display rows and button states, the event for button status change, and a method for scrolling text on the LCD display. To set text to LCD row 0, characters need to be written into a variable at index 0 of the object with the number 6. Communication through this interface consists of simply reading or writing the values of variable tables of other objects with SET and GET functions.

TABLE 1  
VARIABLE LIST OF THE USER INTERFACE ATOMI

UI Atomi		
Object Number	6	
Description	Provides LCD display and button inputs for User Interface purposes.	
void *Variabletable[64]		
Num	type	Description
0	char[32]	Text to LCD row 0
1	char[32]	Text to LCD row 1
2	unsigned char	Button state input / Output for the pins.
3	char[2]	Event message target for button state changed. (Object_Num,Var_index). The defined variable will receive the button state when it has changed.
4	unsigned char	Scroll text on the screen. 0=disable, 1=scroll row 0, 2=scroll row 1, 3= scroll both rows
61	unsigned char	Object Number of this Atomi
62	char[32]	Name of this Atomi
63	unsigned char	Reset.



Figure 4 USB- and IO-Atomi version 2

### 3.3. Active and passive objects

We have divided the object into two categories: *active* and *passive* objects. Active object means that the object can initiate a bus communication sequence i.e. reserve the bus and use it (usually for sending a data request). Passive objects cannot initiate the bus communication sequence, but only decode the address and enable its communication lines on the address match. Active objects usually contain an MCU. Passive objects usually have only some function-related circuitry, such as IO pins, an AD converter or a Bluetooth device, which can be used by an active object.

Originally, all embedded object were active objects. Using both active and passive objects (instead of using only active objects) enables us to create single processor systems with several inexpensive peripheral objects, thus reducing the cost of the system. However, using passive objects has consequences at the software level. As passive objects often do not have a microcontroller, they cannot implement the generic logical interface. Instead, for each passive object there are software drivers which implement the logical interface via proprietary GET and SET functions. This driver must be included in every active object that is intended to access the passive object. Each driver uses its own prefix for the GET or SET function in order to be able to use several passive objects simultaneously. As an exception, there are also passive Atomis that do have a microcontroller. In those cases the Atomis usually implement the generic logical interface and thus do not need a driver to be accessed.

## 4. ATOMI-OBJECTS

Atomi-objects (Figure 4) are horizontally connectable embedded objects. Atomi-objects implement the Atomi II framework [3], which we have defined. It consists of a bus definition, arbitration method, addressing method, physical definitions and rules for expanding the design.

### 4.1. Atomi II Framework

The EOC relies heavily on the interface between the objects, i.e. the physical and logical interconnection. The Atomi II framework is a low-cost, versatile, general purpose bus with some related techniques and physical definitions which will

enable this kind of system architecture. We have studied several existing buses, for example serial, parallel and hybrid buses. According to our research on existing buses, there is no bus standard that has an optimal combination of features for the EOA as such. Also, the existing arbitration methods require too many pins and overly expensive logic for the EOA. For these reasons we have defined a hybrid bus system and developed a proprietary addressing and arbitration method.

The Atomi II framework is a hybrid of existing techniques appended with new addressing and arbitration techniques. The bus is a 9+3 bit parallel bus, called AtomiBus II. To minimize pin usage of the controlling MCU, all except one of the bus lines are shared between general purpose IO and addressing, data, or control functions. The key elements of the bus are the addressing and arbitration methods and analog connections. The bus contains the following I/O lines:

- shared 9-bit general purpose IO/address line (IO)
- shared general purpose IO/set line (SET)
- shared general purpose IO/acknowledge line (ACK)
- address set line (ADDR)

This makes a total of 12 lines. Additionally, the controlling MCU in an active object needs one I/O line for the arbitration logic (BUS\_REQ). Each object does not need to use all the lines of the bus (see Chapter 4.4). The only mandatory line is the ADDR line. In addition to the control IO lines, the bus has the following voltage lines:

- Two ground lines
- Operating voltage line, default 3.3 Volts
- Unregulated direct voltage line, which is meant for higher current devices, such as DC motors. Voltage depends on the power source used.

#### 4.2. Bus reservation and arbitration

The schematic of the arbitration logic is presented in Figure 5. The bus is reserved when any of the control lines (ADDR, SET or ACK) are low. Correspondingly, the bus is free when all control lines (ADDR, SET and ACK) are high. The bus is reserved by enabling the bus reservation logic by setting the BUS\_REQ signal to high impedance (input mode in MCU). When disabled, the BUS\_REQ signal is actively driven low. This logic implements a proprietary daisy chain type arbitration method which is modular, freely expandable and implements a geographical priority scheme.

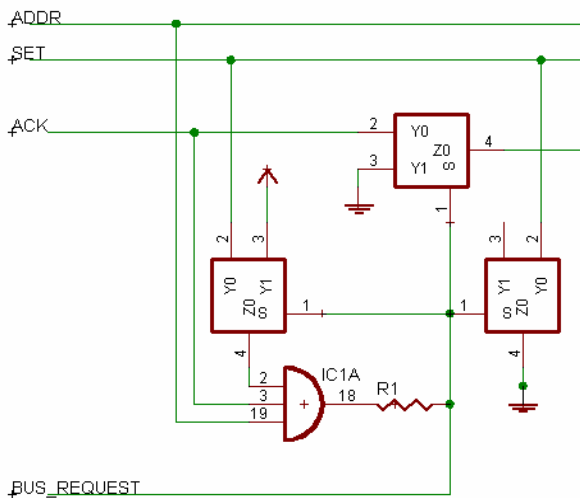


Figure 5 Atomi arbitration schematic

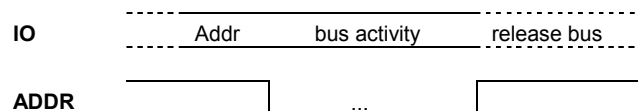


Figure 6 Address selection

#### 4.3. Addressing

The address lines are shared with general purpose IO lines. The address (or object number) must be latched to the 8-bit bus at the beginning of any bus operation by setting the address on the IO lines and lowering the ADDR line as shown in Figure 6. The falling edge of the address line signifies that the address is on the bus. After latching the ADDR line low, the IO pins are free to be used for any purpose. The ADDR line must be kept at logical zero during the whole bus

transaction period, thus reserving the bus and signifying that the addressed object must remain online during this time. What happens in the bus activity phase can be different for different object types. For example, this period can be used to measure the input values of the I/O pins of a passive object, or active objects can transfer data with any protocol, serial or parallel. The address can be either an 8-bit data byte or each object can use one of the IO lines as their own chip select line. Both addressing types are compatible with each other, and they can both be used at the same bus at the same time. The implementation for the addressing is simple and low cost; in its minimal form it can be implemented with only one logic gate, a single D-type latch.

#### **4.4. Pin usage and data transfers**

Due to the protocol-independent addressing and analog IO lines, several different data transfer methods can be used for inter-object communication. Each object is required to be connected to only those bus lines that it uses. Typically the active object or objects that control the system must have a connection to each line in order to access all objects on the bus, but the passive objects or objects with small MCUs use only those lines that are needed for their communications. The only mandatory IO line for each object is the ADDR line. Any of the rest of the lines can act as data transfer and addressing lines. Thus, even a two-pin connection to the bus is possible, when for example a one wire serial data transfer is used, and the chip-select is chosen to be the same IO line that is used for the serial data. Since the bus lines are all analog, analog signaling is also possible within certain limits. The analog signal must not exceed the limits of 0 volts and the operating voltage, nor exceed the current limit of the bus connectors or track width. We have tested the analog connection by connecting a key matrix to the IO pins. Furthermore, since there are 12 IO lines available for use, several electronic modules, for example the popular HD44780 LCD-display, can be used through the bus by merely adding the latch gate for address recognition. Furthermore, this system is arbitrarily scalable, IO lines are not tied to any specific protocol, and implementation is simple and inexpensive.

#### **4.5. Physical definitions**

The basic size of an Atomi is 48 mm x 14 mm. The 48 mm width is fixed but the 14 mm length can be expanded if necessary. Each board connects to the others so that the long sides are parallel to each other. The bus runs through each board via bus connectors. The board size is optimized according the size of a euro PCB panel, which is often used in production. By connecting the boards in parallel, the Atomi-objects form a prototype of a PCB layout that can be produced as a single PCB. This has also been considered in the bus connector selection. The bus connectors fit directly to the bus tracks without a separate footprint or tracks leading to the footprint. Thus, the layout of Atomi-objects in an application can be manufactured as a single board by copying the layouts together as such based on their positioning in the prototype.

### **5. TELEPRESENCE ROBOT**

The telepresence robot, or telerobot, is a test case of a complex system to be built with the embedded objects. The telepresence robot system consists of a two-wheeled, human height robot and its computer counterpart. The robot contains video and audio exchange capability via a WLAN connection and a controlling and balancing system for driving it with two wheels. The robot is built in two versions. The first version consists of a laptop PC and the Atomi-objects. This version demonstrates the possibility of using the EOC together with a computer. The second version consists of only Atomi-objects, demonstrating the flexibility of the EOC for modifications and proving the capabilities of the EOC by replacing a computer. The computer counterpart, which is used to remote control the robot, is a regular PC with audio and video capabilities running with a robot control application.

The purpose of this test case is to validate the suitability of the EOC for relatively complex systems. The telepresence robot consists of several concurrently operating functions, which all are more or less complicated. Thus it is a good project for testing our concept. The test case aims to validate several EOC-related issues: the robot implements the Embedded Object Architecture (EOA) with the Atomi II –framework, the architecture of the robot is designed with an object-oriented design method that is modified for the EOC, and the robot is built using incremental device development, which is made possible by the architecture. The robot system will be analyzed for its functionality, design time and overall costs with respect to a similar system made with computers or a system made as an embedded system without the EOC. The following chapters will present the architecture and technical details of this robot and the current status of the test case.

### 5.1. Telerobot – PC version

The object diagram of the first telerobot version is shown in Figure 7. It is built around a PC that has video and audio capabilities and an USB port for connecting it to Atomi-objects. In this diagram the composition symbol is used to depict built-in components of the PC. The containment symbol means the attachment of an external component to an object. The stereotype property notates the hardware platform on which a class resides. The associations, aggregations and compositions between Atomi-objects mark bus connections.

The upper part of the diagram shows the PC and its peripherals. The PC software runs Skype [13] for audio and video connection between the robot and the remote PC. For driving the robot, the PC runs a simple Atomi packet relay, which relays Atomi message packets between the Movement-object and remote PC. The remote PC runs software which translates key presses into driving commands. The robot is driven with the arrow keys of a standard PC keyboard. The remote PC software sends the driving commands continuously to the robot as UDP packets.

The lower part of the diagram shows the Atomi construction. There are four Atomi-objects: two DC Motor-Atomis, an ADC-Atomi and an USB-Atomi. The DC Motor-Atomi controls a DC motor and reads its encoders. Its default software sets the desired speed for a motor using a PID control and the encoder feedback. The DC Motor-Atomi is used here with its default software with geared DC Motors connected to them. An ADC-Atomi contains eight inputs that can be used as digital inputs or analog inputs to an AD converter. The ADC-Atomi is here also used with its default software. It has four bumper switches and an analog tilt sensor connected to it. The USB-Atomi has a separate USB port for connecting to the PC. Here it acts as a platform for the Movement-object. The Movement-object controls the balancing and driving functions of the robot. It checks the pose of the robot and the received commands ten times per second, and controls the DC motors accordingly. It also has a timer function, which stops the robot if commands do not arrive in a certain time period, for example in case of network congestion or other problems in the driving. The Movement-object exposes a separate variable list interface to the USB port, which is specifically used for getting driving commands from the PC. Thus, the Atomi construction forms a new object from a group of objects with encapsulated functionality and a corresponding separate interface.

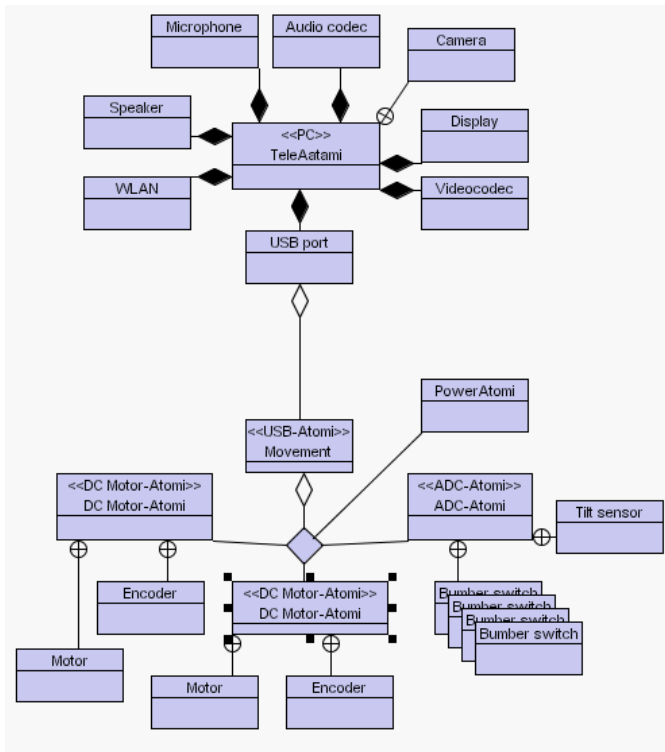


Figure 7 Telerobot version 1 object diagram

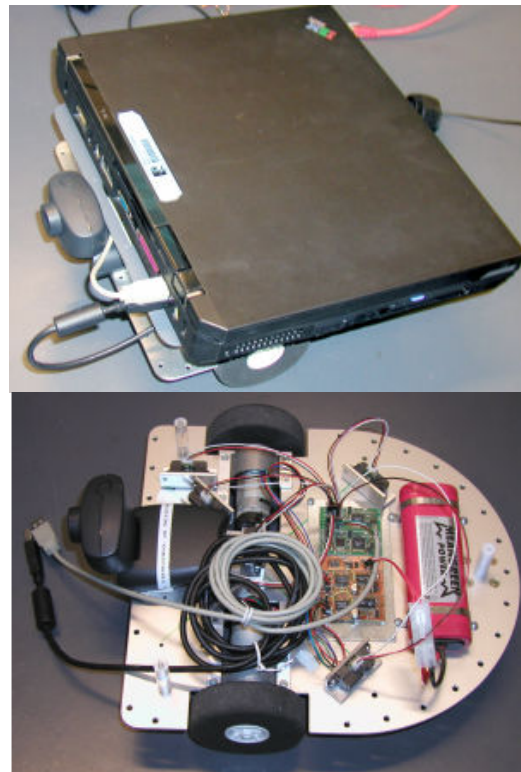


Figure 8 Telerobot backup in driving form (top) and opened to show the electronics (bottom)

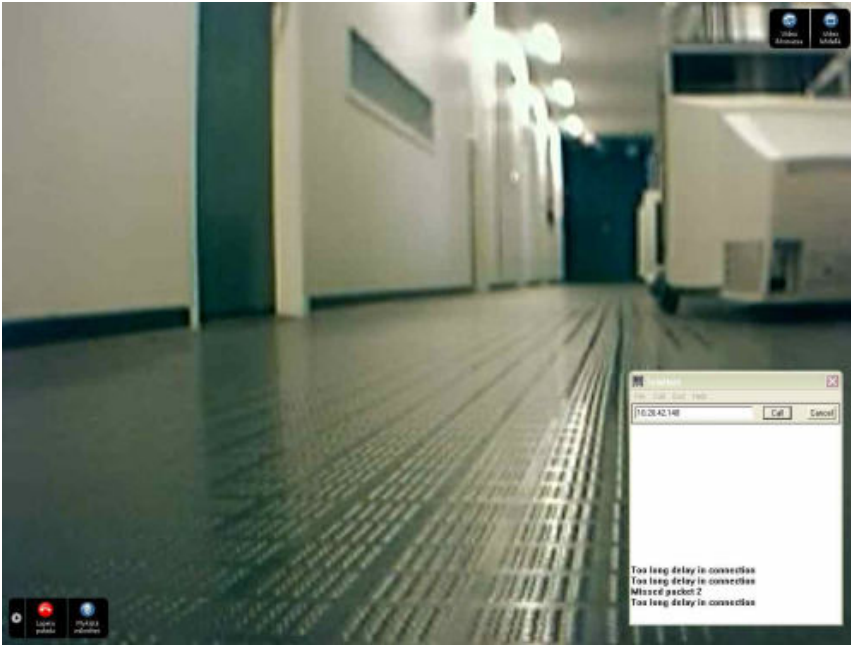


Figure 9 (left) View of remote PC driving the telerobot version 1 on the hallway of our laboratory. The Skype video screen is at full screen view and the driving command window is at low right corner.

Figure 10 (right) Two-wheel standing robot base

### 5.1.1. PC version results

The PC version of the telerobot was built and tested. The video and audio part of the telerobot system is based on Skype, which in good network conditions produces nice video and sound quality with a reasonably short delay and thus is very suitable for this kind of system. The driving software architecture is very simple and works very nicely. The driving packets are short and quickly delivered. Overall, the PC parts are simple and functional.

The Movement-object is a more complicated construction as it consists of several interoperating Atomi-objects. The Atomi-objects are powered by an 8.4V battery pack, which is connected to a Power-Atomi. The Power-Atomi regulates a 3.3V operating voltage from the battery voltage for the rest of the Atomis, and distributes the direct battery voltage to the unregulated direct voltage line for the DC Motors. The Power-Atomi has a thermal protection for overload currents.

The main software of the Movement-object resides in an USB-Atomi, which is an active object. The USB-Atomi is based on an USB chip by FTDIchip and an Atmel AVR ATmega16 MCU, where the main software of the object resides. It receives the driving commands through the USB interface and operates the DC Motors and ADC-Atomi accordingly in a continuous loop. The DC Motor-Atomis and the ADC-Atomi are operated through the Atomi bus interface. The timeout function in the Movement-object kicks in if the packets have not arrived within 0.5-second intervals, stopping the robot and preventing it from driving to obscure places.

The DC Motor-Atomi is based on an A3953 motor driver by allegro Microsystems and Atmel AVR ATTiny2313 MCU. It is capable of driving motors with up to 50V. Its current is limited to 1A. The DC Motor is implemented as a passive object even though it has a microcontroller. This is because it does not have any events and thus does not need to reserve the bus. The motors connected to the DC Motor-Atomis are Micromotors RH 158-12-30 with a hall effect encoder.

The ADC-Atomi is basically an ATmega 16 MCU with its internal 10-bit SAR type AD-converter used for conversion. It is capable of measuring approximately 100k samples per second divided by the number of channels in use. It is an active object and its pins can be used either for AD conversion or for generic IO. In the telerobot plan the AD converter reads the tilt sensor output. In the actual implemented telerobot the pins are connected to 6 infrared distance sensors.

The robot base is shown in Figure 10. It is made of aluminum and steel. It is approximately 75 cm tall and weighs less than 1 kg. It has a place for Atomi-objects in the bottom, and a red 12V battery pack above the Atomi-objects. At its top

there is a place for a PDA or small PC, or a display device. In case of a failure in balancing the robot is prevented from falling by a steel loop. The loop bends a little, and thus softens the fall without harming the electronics.

In general, the electronics and the Atomi II framework work well. However, we had problems with our balancing algorithm for the two-wheel standing robot. With our current PID based algorithm the robot can stay in balance, but it drifts either forward or backward. Thus, we weren't able to implement the steering commands to the robot with enough confidence to put the laptop aboard. Also, the backlash of the wheels causes the algorithm to switch the driving direction back and forth excessively, thus consuming quite a lot of energy in vain. Anyway, we are confident that these problems will be solved with suitable state space controlling equations and a low pass filter for the backlash problem, as reported in [14]. For this article, however, we didn't have time to implement those, but chose to test the Atomi-objects with a regular two-wheel + a castor wheel robot base as shown in Figure 8, instead of the original standing two-wheel robot base shown in Figure 10.

With the configuration shown in Figure 8, we did some test drives along the corridors of our laboratory. The system was very functional and the video feed tolerable (Figure 9). The network delay poses the biggest problems for driving, which was expected. There was a significant difference between driving with a congested network and a non-congested one. We tested driving both during the daytime, when everyone is at work and using WLAN, and in the evening, when very few people are using the WLAN. During the daytime the video was very slow with a long delay, making driving nearly impossible. The robot could be driven only in short distances, and it was necessary to wait and see where the robot stopped between each journey. In the evening, driving was much easier due to available WLAN band, and the robot could be driven in a constant motion. Some extra effect on the driving experience also caused the responsiveness of the PID control in the Movement-object, which was fine-tuned a bit quicker after a few test drives. But in terms of the EOC and computer co-operation, the test shows the design and implementation methods feasible and the Atomi II framework functional

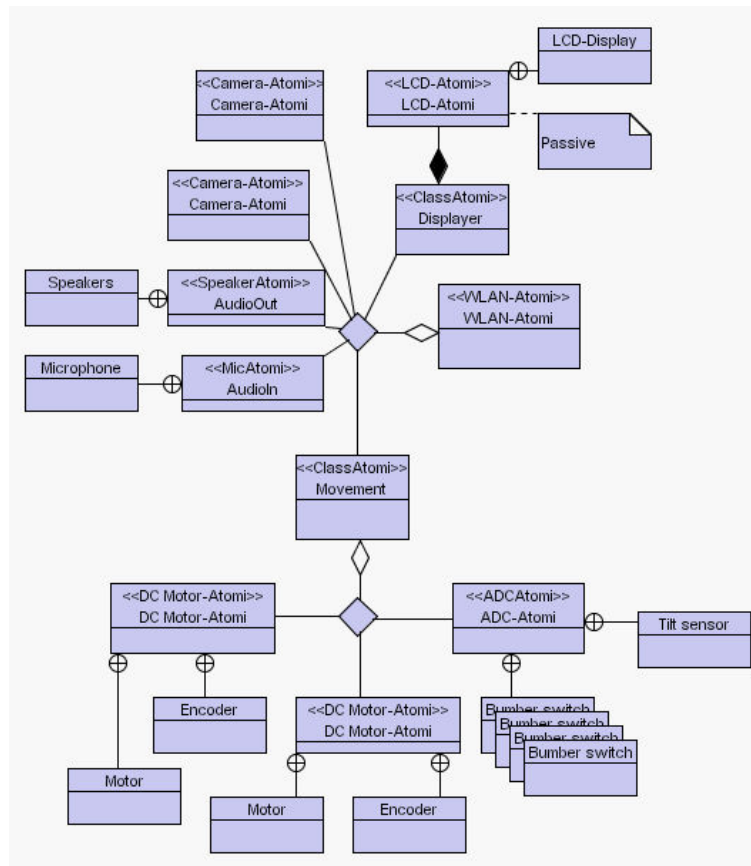


Figure 11 Telerobot version 2 object diagram

## 5.2. Telerobot – Atomi version

Figure 11 shows the object diagram of the telerobot that is built completely with Atomi-objects. In this diagram the composition is used on an object that acts as a master to passive objects. As the aggregation symbols show, the WLAN-Atomi is the master object of this system. WLAN-Atomi has a TCP/UDP/IP stack and a WLAN connection. It receives data from the controlling computer via a WLAN connection by UDP protocol, and forwards the data to different objects according to the UDP port assignments. Each object has its own ports. The WLAN-Atomi feeds Displayer, AudioIn and Movement-objects with the data it receives from the computer. Similarly, it sends data to a remote PC from Camera-, AudioOut- and Movement-objects.

The Displayer-object decodes the MJPEG-video data received from the PC and shows it on the LCD display via LCD-Atomi. The Camera object correspondingly reads MJPEG video from the CMOS Camera and feeds it to the WLAN-Atomi. The AudioOut-object decodes the iLBC (internet Low Bitrate Codec) [15] audio feed and plays it through speakers. The AudioIn-object correspondingly encodes the microphone signal with iLBC and sends it to the WLAN-Atomi. The Movement-object is the same as before, but the USB bus is switched here to the normal Atomi bus. The PC software for this version is the same as for version 1, except the Skype part will be changed to proprietary software.

### 5.2.1. Atomi version results

Currently the Atomi version of the robot is incomplete, but several functional Atomi-objects have been created and tested. The robot balance algorithm problem also concerns this version of the telerobot, so the two-wheel standing robot has not been used. The Movement-object created for the first telerobot version is almost directly usable with this robot version, only the bus interface software is changed from a USB to generic Atomi bus. The remote PC software is also used in this version, but will be appended with our own video and audio feeds. Furthermore, there are WLAN- and Camera-Atomis ready for this robot. And, according to the incremental device development principle of the EOC, we shall continue developing the robot piece by piece.

The Camera-Atomi (Figure 12) is a small 25 x 48 mm Atomi. It is an active Atomi consisting of ARM7 MCU by Atmel and a small CMOS camera by TransChip. The camera is capable of generating a VGA resolution (640x480 pixels) MJPEG video with 25 frames per second. It runs at 3.3 volts and consumes 75 mA when running at its maximum speed. The camera can also run at lower frame rates and take single shots, consuming less power accordingly. It can also take video or still images with different formats such as several RGB formats, YUV422 and YCrCb422. It also has adjustable contrast, brightness, saturation, hue, gamma, resolution and MJPEG compression quality.

The Camera-Atomi has a minimal number of components. It does not have a FIFO buffer or external oscillator. Instead, the data bursts from the camera are caught with an interrupt routine in the MCU with the help of a special clock synchronization circuit. The circuit slows down the clock of the camera at the moment of the data burst, enabling the MCU to catch it. Thus the Camera-Atomi can write the image data directly to any Atomi simultaneously while it is being received from the camera. In the telerobot, it is written directly to a transmit buffer of the WLAN-Atomi to be sent to the remote PC with minimum delay. The Camera-Atomi has been tested with the USB-Atomi and WLAN-Atomi. We have written a DirectShow [16] driver for PC to access this camera from both USB and TCP/IP interfaces. The DirectShow driver enables us to use any DirectShow filters that are available for Windows. Thus, for the stereo image we use the Stereo Transformation Filter [17] by 3dtv.at, which enables us to view 3D video with several different display methods.

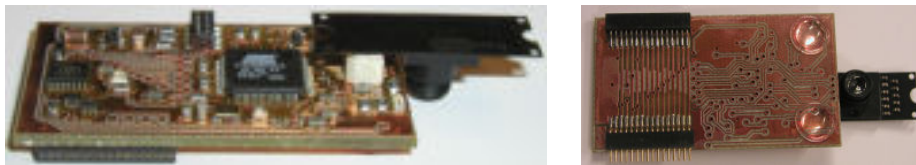


Figure 12 Camera-Atomi

As mentioned before, the camera has adjustable image compression quality. With different quality settings the single VGA resolution frame size can vary between 16 and 100 kilobytes. At a tolerable image quality the data rate is approximately 20 - 30 kilobytes per frame. This presents a tight requirement for the WLAN-Atomi to be able to forward enough data in time. In the telerobot, there will be two Camera-Atomis onboard. For these only the data rate of the WLAN-Atomi should be 2 cameras x 25 kB/frame x 25 fps = 1250kB/sec, which is equal to 10Mb/second i.e. almost the limit (11Mbit/sec) for a typical WLAN based on IEEE802.11b. However, the image quality of the video can be

altered while running, which enables dynamic switching between a faster frame rate and better image quality. In other words, the robot can use a higher frame rate and low image quality when the robot moves fast and the fast frame rate is required. Correspondingly, the robot can employ a low frame rate but accurate image in slow speeds or in standstill, when accurate image of people present or the scene is needed. This is an interesting option to test in the future.

The WLAN-Atomi has also been developed and tested, but not in its final form. As a quick setup we have created an Atomi out of an Ethernet board and added a WLAN bridge to its Ethernet socket. The Ethernet board has an ARM7 MCU with the uIP [18] TCP/IP stack running in it. While the Ethernet and the WLAN bridge are only a temporary setup, the ARM7 and the stack side will be part of the final WLAN-Atomi. The WLAN-Atomi exposes transmit and receive buffers to the bus for each Atomi. The transmit buffers are sent as soon as something is written to them in order to minimize delay. The speed of this Atomi is crucial for the functionality of the whole system. Currently, we have been able to make the uIP stack to send 320 kilobytes of data in a second into one UDP connection by running the ARM processor at 60MHz. If the Atomi reads data from the Atomibus simultaneously while sending data to the UDP connection, the data rate drops by a half. The data rate is barely enough for tests, but we believe it can still be increased by optimizing the stack. Currently we have tested the WLAN-Atomi by sending through one MJPEG video feed. Further tests will be made with two-directional simultaneous data transfers and multiple sockets.

### 5.3. Evaluation of the EOC

Even though our project is unfinished and it has been delayed, it validates some issues of the EOC. The robot has been designed with an object-oriented design method and modeled with UML. The architecture consists of objects, either Atomi-objects, or objects consisting of several Atomi-objects. In our robot the Movement-object is an object consisting of several objects. The USB-object is a class type of object: it treats the DC Motor- and ADC-Atomis as private objects, and exposes separate public variables to the USB interface. As a whole, the telerobot is built from several other objects in addition to the Movement-object. The robot shows that such software-like architecture can be realized with the EOC, although there are not conclusive tests yet to prove the whole design is functional.

The robot has been built in an incremental fashion from the start. First we created the USB-Atomi, tested it, and moved to the ADC-Atomi, then to the DC Motor-Atomi, Camera-Atomi and most recently the WLAN-Atomi. Each Atomi is individually made and tested, and then added to the existing robot. The incremental development is possible due to the common interface between objects, which validates this part of the Atomi II framework.

Current data rates have not yet presented a problem for the Atomi bus. The fastest tests have been made with 1MB/sec between two microcontrollers using a parallel data transfer method. The bus is terminated by the diode termination of MCU IO pins, which clamps both over and undershoot on the data lines at about 0.3V over operating voltage or below ground. Specific tests for high data rate limits have not been conducted.

The design time for the current system is approximately 2 work months for the robot version 1 and 2 work months continuation for the second version. We estimate it to be the same whether the system was built with the traditional method or the EOC because all the Atomis have been created from scratch. However, the real benefit of this system would be evident if the modules were already made for example for some other devices, and could be reused here. This actually is the case in our laboratory for some other projects: for example USB- and ADC-Atomis are now in use in several projects, combined to a selection of other Atomi-objects [19]. The USB-object is also in frequent use when testing new Atomi-objects that are under development. But because the telerobot project is the pioneering one, the actual design time benefits have unfortunately not yet affected this project.

The general experience of the EOC and Atomi-objects is that this framework provides easy modifications, virtually unlimited expansions to existing designs, and most importantly, the reuse of existing objects. The reuse of objects and unified logical interface are important for the easy-to-use aspect. When there are some more Atomi-objects available, new devices can be constructed out of ready-made objects, in which case the designer is not required to have any expertise on PCB design, component sourcing, assembling a PCB or other things related to electronics production. Due to the unified logical interface, learning how to use new objects becomes easier. The user only needs to determine what the object does and what each variable stands for; the physical connectivity or data representation methods do not need to be studied. These properties together allow for a short design time for new systems. Additionally, the architecture enables an incremental design and testing process. A prototype of a system can be built piece by piece, as new objects can be attached to an existing system without recreating the whole system. Additionally, a final product of a prototype built with ready-made Atomi-objects can now be manufactured as one PCB with the same layout as in the prototype. Thus, the electrical properties of the prototype and final product version are very close to each other. This minimizes the risk and costs of starting a production of a new device.

The telerobot project is still unfinished, which kept us from being able to validate and analyze some points. However, the fact that this project has been delayed shows how slow the making of electronics can be, and at the same time it reinforces the need for Lego-type technology like the Atomi-objects. If these objects were ready and tested, the building would have been done very quickly. If we look to the future, we believe that there is a need for the EOC. As devices become more and more complex, simple principles are needed to enable designing and maintaining of devices. Success in validating the EOC for embedded system communities could inspire IC manufacturers to develop Atomi-like packaging and interfacing for integrated circuits. This kind of a trend can already be seen in the electronics industry, as more integrated and easy-to-use modules come to the market all the time (for example [20]), and the programmability of chips increase (for example [21]). Only the common interface is missing, and that is what makes Atomi-objects feasible.

## 6. CONCLUSION

This paper presented the Embedded Object Concept (EOC) and a telepresence robot system which is a test case for the EOC. The paper presented the motivation and the general architecture for the EOC, and the latest version of our embedded object implementation, the Atomi-objects. The Atomi-object is made by our Atomi II framework specification, which is being tested in a relatively complex test case. The test case presented in this paper is a telepresence robot consisting of a two-wheeled human height robot and its computer counterpart. The robot has been constructed with Atomi-objects using incremental device development, which is made possible by the architecture of the EOC. The robot is made in two versions. The first one is completed and tested. The second one is still incomplete, but most of it has been successfully tested. The EOC will save a considerable amount of development time if there are any ready made objects available. Present experiences with the EOC and embedded objects are very promising.

## ACKNOWLEDGMENTS

This research was partially funded by Infotech Oulu Graduate School, the Finnish Academy and University of Oulu, Robotics Group.

## REFERENCES

1. A. Berger: *Embedded Systems Design – An Introduction to Processes, Tools, and Techniques*, CMP Books, Lawrence, Kansas, USA, 2002.
2. T. Vallius, J. Haverinen and J. Röning: *Object-Oriented Embedded System Development Method for Easy and Fast Prototyping*, International Conference on Machine Automation 2004 Nov 24- Nov 26, Osaka, Japan.
3. T. Vallius and J. Röning: *ATOMI II - Framework for easy building of object oriented embedded systems*, 9th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, Cavtat, Croatia, August 30th – September 1st, 2006.
4. M. Edwards and P. Green: *An Object-oriented Design Method for Reconfigurable Computing Systems*, Design, Automation and Test in Europe Conference and Exhibition 2000. Proceedings, 27-30 March 2000, Page(s): 692 - 696.
5. E. Grimpe and F. Oppenheimer: *Object-oriented high level synthesis based on SystemC*, Electronics, Circuits and Systems, 2001. ICECS 2001. The 8th IEEE International Conference on, Volume: 1, 2-5 Sept. 2001, Page(s): 529 - 534 vol.1.
6. W. Nebel and G. Schumacher: *Object-oriented hardware modelling-where to apply and what are the objects?*, Design Automation Conference, 1996, with EURO-VHDL '96 and Exhibition, Proceedings EURO-DAC '96, European, 16-20 Sept. 1996, Page(s): 428 -433.
7. *Object-oriented programmable integrated circuits*, <http://www.oopic.com> [Accessed 6.7.2006]
8. *The RoboBricks Project*, <http://www.gramlich.net/projects/robobricks/> [Accessed 6.7.2006]
9. *The SimmSticks*, <http://www.simmstick.com/> [Accessed 6.7.2006]
10. T. Gorton: *Tangible Toolkits for Reflective Systems Modeling*. Masters of Engineering Thesis. Massachusetts Institute of Technology. May 21, 2003.
11. The Tower System Web-pages, <http://gig.media.mit.edu/projects/tower/> [Accessed 6.7.2006]
12. J. Martin and J.J. Odell: *Object-Oriented Analysis and Design*, Prentice Hall, 1992.
13. Skype, <http://www.skype.com> [Accessed 6.7.2006]
14. F. Grasser, a. D'Arrigo, S. Colombi, A.C. Rufer: *JOE: a mobile, inverted pendulum*, Industrial Electronics, IEEE Transactions on Volume 49, Issue 1, Feb. 2002 Page(s): 107 – 114.

15. *Internet Low Bitrate Codec*, <http://www.ilbcfreeware.org/> [Accessed 6.7.2006]
16. Microsoft DirectShow, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directshow/htm/directshow.asp> [Accessed 6.7.2006]
17. StereoTransformationFilter at by 3dtv.at, <http://www.3dtv.at/> [Accessed 6.7.2006]
18. The uIP Embedded TCP/IP Stack, <http://www.sics.se/~adam/uip/> [Accessed 6.7.2006]
19. A. Tikanmäki, T. Vallius and J. Röning: *Qutie – Modular methods for building complex mechatronics systems*, International Conference on Machine Automation (ICMA2004), 24-26 Nov 2004, Osaka, Japan.
20. Digi Connect<sup>®</sup> Wi-ME, <http://www.digi.com/products/embeddedmodules/digiconnectwime.jsp> [Accessed 6.7.2006]
21. PSoC Mixed-Signal Controllers, <http://www.cypress.com/>, [Accessed 6.7.2006]