國立交通大學

電機資訊國際學位學程

碩士論文

Integration and interoperability in the context of enabling end-user to incorporate arbitrary electronic devices and software into a collective population of resources capable of cooperation.

研究生：**Krzysztof Kamil Jacewicz,** 秦學思

指導教授：黃經堯 博士

中華民國一〇二年一月

Integration and interoperability in the context of enabling end-user to incorporate arbitrary electronic devices and software into a collective population of resources capable of cooperation.

研究生：秦學思　　Student：Krzysztof K. Jacewicz

指導教授：黃經堯 博士　　Advisor：Dr. ChingYao Huang

國立交通大學

電機資訊國際學位學程

碩士論文

A Thesis

Submitted to EECS International Graduate Program

National Chiao Tung University

in Partial Fulfilment of the Requirements

for the Degree of

Master

January 2013

Hsinchu, Taiwan, Republic of China

中華民國一〇二年一月

National Chiao Tung University

Authorization of Copyright to E-version of Doctoral/Master's Thesis

(For licensor to bind as 2nd page of the thesis's inside cover)

The degree thesis authorized is required by _____ (Dept.), National Chiao Tung University in _____(Month), _____(Year).

Thesis Title:
Integration and interoperability in the context of enabling end-user to incorporate arbitrary electronic devices and software into a collective population of resources capable of cooperation.

Advisor: Dr ChingYao Huang, 黃經堯 博士

■ Agree □ Disagree

I hereby, non-exclusively and for free, authorize this thesis to National Chiao Tung University and University System of Taiwan Library. Based on promoting the idea of 'Resource Sharing & Mutual Benefit and Collaboration' among readers, as well as feed back to the society and academic research, National Chiao Tung University and UST Library are allowed to include, reproduce and utilize in different forms (paper, CD-ROM, other digitized materials) regardless of regions, time, and frequency. Abiding by the Copyright Law, readers can search online, read, download, or print out the thesis.

Range and Time for Uploading:	
NCTU and UST LAN	■ Accessible from /Y. /M. /D.
Off-Campus Internet	■ Accessible from /Y. /M. /D.

Authorized by:

Signature: _____

/Year /Month /Date

National Chiao Tung University

Authorization of Copyright to Doctoral/Master's Thesis

(For licensor to bind as 2nd page of the thesis's inside cover)

The degree thesis authorized is required by _____ (Dept.), National Chiao Tung University in _____(Month), _____(Year).

Thesis Title:
Integration and interoperability in the context of enabling end-user to incorporate arbitrary electronic devices and software into a collective population of resources capable of cooperation.

Advisor: Dr ChingYao Huang, 黃經堯 博士

■ Agree

I hereby, non-exclusively and for free, authorize this thesis to National Chiao Tung University. Based on promoting the idea of 'Resource Sharing & Mutual Benefit and Collaboration' among readers, as well as feedback to society and academic research, National Chiao Tung University is allowed to include, reproduce and utilize in paper. Abiding by the Copyright Law, readers can read or print out the thesis.

The thesis accounts for one submitted by me to the MOEA's Intellectual Property Office for the patent application (Ignore if not applying). The application number is _____, please postpone the disclosure of the thesis to ____/Year____/Month____/Date.

Authorized by:

Signature: _____

/Year /Month /Date

Authorization of Online Access to Doctoral/Master's Thesis, National Central Library

ID: GT00 (+ Student Number)

The degree thesis authorized is required by _____ (Dept.), National Chiao Tung University in _____(Month), _____(Year).

Thesis Title:
Integration and interoperability in the context of enabling end-user to incorporate arbitrary electronic devices and software into a collective population of resources capable of cooperation.

Advisor: Dr ChingYao Huang, 黃經堯 博士

I hereby agree to non-exclusively and for free authorize this thesis (Abstract included) to National Central Library, who is allowed to, regardless to regions, time, and frequency, reproduce in different forms (MF, CD-ROM, other digitized products) and upload to the Internet for readers to non-profit-making search online, read, download, or print out the thesis.

※ The non-profit-making purpose of online search, reading, downloading, or printing should follow the copyright laws and regulations.

Authorized by:
Signature: _____

/Year /Month /Date

NATIONAL CHIAO TUNG UNIVERSITY
EECS International Institute
Master's thesis
Krzysztof Kamil Jacewicz

20.01.2013

ABSTRACT

Objectives of the study

How to enhance End-users' experience by developing a system (and a standard) for integrating owned devices and programs into a network of resources, and be able to selectively and remotely access functions distributed across the network? And how to enable developers to easily make their existing products compatible with the system?

Academic background and methodology

Author's academic background is in EECS with the concentration in Networking and Telecommunication, and additional interest in Compiler Theory and programming for embedded systems. Professional background includes work at Rawspeed (Beijing) Technology Ltd., as Product Manager for CDN related products, at Vale (Beijing) Internet Technology Ltd., as Product Manager web service and user experience, at Rulingcom Digital Inc. (Hsinchu) as Project Manager, working on CMS system, GIGABYTE Technology (Xindian), working as Account Manager for Channel sales division GIGAZONE, and at American Megatrends Inc. Taiwan Branch (Taipei), as Marketing Specialist and supporting engineer working on projects directly related to cloud computing, remote server management, and web service architecture. Methodology used in the process of research includes review of publications and knowledge acquired during professional work.

Findings and conclusions

In process of the research architecture has been designed and proposed for integration of hardware and software resources. Key finding was that it can be implemented, and even used in a commercial project with much success. It was also found that created solution has saved time and complexity when adding RPC functionality to an existing system and did not require the existing code (originally the eventuality of incorporating ignoring RPC support) to be redesigned in any bit. As a result, the system was found to enable SOA as a Service

Keywords : RPC, Middleware, M2M communications, Internet of Things, Home automation

Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisor Prof. ChingYao Huang for the continuous support of my study and research, for his immense patience, his motivation, enthusiasm, and given trust. His guidance helped me in all the time of research and writing of this thesis, with lot of understanding received from his side.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Kai-ten Feng, and Dr. Sheau-Ling Hsieh, for their encouragement, insightful comments, and hard questions.

I want to add special thanks to Allen Chang, the founder and owner of Rulingcom Digital Inc., where I have worked when I first came to Taiwan. Allen himself encouraged me to follow master degree at the National Chiao Tung University (he is the alumni himself as well) and gave me the flexibility needed for working and studying both in full-time mode.

Last but not the least, I would like to thank my family: my parents Ryszard and Wanda, for supporting me spiritually throughout my life.

Table of context

List of abbreviations

API	= Application Programming Interface
IOT	= Internet Of Things
EUD	= End User Development
M2M	= Machine-to-machine (Communication)
RPC	= Remote Procedure Call
RMI	= Remote Method Invocation
SI	= System Integrator

List of appendices

1 Introduction: context to the thesis study

This chapter will selectively highlight few areas of interest significantly related to the matter of this paper. Before the actual problem is defined in the next chapter, here is a brief introduction to the greater context from which the study has begun.

1.1 Mobile computing

One of the earliest origins of the idea for this research was my first study of mobile computing after I got my first ever mobile device – a HP Jornada handheld PC running on Windows CE (now discontinued). I had a special interest in developing my own software to run on the mobile device, and later, another strong interest in remotely communicating from the mobile device to the PC and controlling some features of the server application. My first finding was, that it was much more complex to develop for mobile, due to things like SDK, cross-compiling, emulating, debugging on a device, and few others. Also, there were no tools for native development on the mobile device at the time. Even now in 2013 this is still not common. I am using Pepe Le Compiler on my Android handset to natively compile Pascal code on the device and run it.

My second finding was that it was quite time consuming to implement remote communication between my mobile device and my PC. These were two different platforms, and I had to re-learn socket programming in another programming language, that was supported in the SDK for the mobile device. As an effect, I had to write conceptually the same code twice, in two programming languages (at least). These two findings have eventually made me more appreciative toward web programming. As long as I wanted to use my Smartphone as a remote control for a PC not the other way around, using web browser on the phone was much easier than

going through development of a native (or Java) application. Later, with the arrival of HTML5 and some support for accessing in-phone API via webAPI, I started to appreciate HTTP protocol even more.

1.2 Cloud computing

My next important study that helped me to approach the research covered here, was in Cloud Computing. The theory that this study covers is very wide, but my special interest was in web service architecture and user experience.

Web API concept felt in sync with my approach to solve some problems in mobile computing. And as soon after first public cloud services started to be available to end users, I immediately saw an opportunity to help my programming to incorporate these to save lot of coding on information sharing between applications in a network. Cloud storage has rapidly become one of my favorite cloud services. Before the era of Dropbox, I would either need to remotely connect to a database (which was sometimes too much overhead for small apps) or to implement file transfer functionality. Today, I could re-write lots of my old applications in a fraction of time originally spent on them, as I would totally ignore the portion related to data transfer over network. One of my most recent commercial projects was a digital signage player. The player would keep digital photographs in a local folder, and play them on the screen in an infinite loop. If, at any time, the content of the directory changed, the player would list all the files again and restart the loop. No network programming included. However, my customer hoped, he could manage the content remotely from PC in his home, or even better, via mobile phone. He even requested a feature of being able to upload a photo taken with his mobile phone directly to any Digital Sign in his network. I have Upgraded my player without actually upgrading it. I enabled requested functionality to my customer

without changing a line of code in my project. I have set up a Dropbox account for my customer, and installed a client both on his Smartphone and each PC running the player. I have synced the folder with photos on every player with Dropbox. And that's it! My customer can take a photo with his mobile phone, and upload it to Dropbox. The player on the PC will see that the content of the local folder changed as soon as Dropbox syncs it with the cloud. And few moments later, the uploaded photo will display on the Digital Sign. That is my idea of so called "Integration" – customizing systems to one's needs, with minimum effort, by integrating resources designed to share functionality between each other.

On the user experience side, that I mentioned was my other major interest, I found it almost beautiful, how transparent the whole network programming can be to the end user. I am a huge fan of simple, intuitive interfaces that hide all the complexity from the user's experience.

1.3 "… as a Service"

When I was working for American Trends, I was assign to work on a business plan with my colleague with MBA background. The business plan consisted of lot of technology-related problems to address. A lot of it was related to cloud computing, and particularly to web services, and SaaS (Software-as-a-Service) business model. Soon, also IaaS (Infrastructure-as-a-Service), PaaS (Platform…), NaaS (Network…) and all other "aaS'es" came after. Other than the fact that these are all very hot topics on the market these days, it helped my view on the integration problem to evolve to a whole new level. My primary programming language was always Pascal. After I started developing for mobile devices, I had to learn few more programming languages. Even my rusty C++ ability was of no use (except for developing native ARM executables, in which case either Pascal or C++ were both fine). I have learnt

a bit of Java, and Java-based languages, a bit of objective C used for developing for iOS, and learnt few different SDK's (WindowsCE, Android, iOS, Symbian, BREW), programming environments/editors, and emulators. There were quite a lot of new things to learn each time before I was able to start coding, and then some more way to go before being able to compile, and a bit more way to go before I was able to run. The "…as a Service" approach has changed my life for better! Eventually I came up with my own, alternative way of programming for mobile: if I needed to control an asset on my mobile device, I looked up snippets of code in the Internet to learn the reading from and writing to the asset and enclose it in few function calls. Then I use previously developed generic code, to export these few functions as a web service. Now I can immediately use them from any application. Instead of continue to use programming language I am not fluent in, I use Pascal to write an application calling the web API. Or I use HTML and Javascript to create a static page, with an interface to call the web API. And one of my most original ways to use this approach, is to design and put a static HTML page on my phone, that I will use to control assets on my phone via phone's web browser, rather than via mobile app. If I want to change the program, I only need to change the HTML and Javascript. No compiling needed. And I am not claiming that this way is optimal for local on-device use, but it is simpler for me than programming in, say, Java, and gives me lot of additional benefits:

- such code is web-service oriented and thus my app can have much broader context of use
- Once exported functions controlling on-device assets, I am no more bound to use standard and troublesome software development process for Android: no more eclipse, no more compiling, no more app delivering to the phone.

At this stage a concept of the later research was already slowly emerging in my mind. I learnt to see how much space there is to come up with new, alternative ways of doing things. And I started to understand that integrating components into a bigger system saves lot of time otherwise spent on re-inventing the wheel.

1.4 Internet Of Things (IOT)

The first time I really got to know about the IOT was when my thesis adviser Prof. Huang assigned me to work with one of his research teams. Initially I was seeking a subject for research around sensor networks. I thought that I might use semantic web approach to solve some problems in the area. The Semantic Web was a topic I have studies a bit while working at Rulingcom Digital Inc., and at the time it was because everyone on the market was discussing Web 2.0 and we wanted to go ahead into the future, looking for some inspiration for later RND. I thought that Semantic Web can also help in addressing some of the problems of the sensor networks, related to Data interoperability. Little did I know, that Semantic Web gives much more potential help than only dealing with data. My study into IOT helped me to see, that there is a huge potential in using IOT to integrate devices and programs. I found out, that developing for the IOT was very challenging and complex problem. Lot of tools was still under development, or only existed as reference designs. Lot of the topics was still in the stage of theory. But going through lot of work, IOT was offering a chance for ending up with a solution giving super-intuitive, simplified and interactive user experience. That was just what I was always after – hiding all the complexity from the user experience and providing him with pure functionality, dressed in an intuitive interface.

1.5 Machine-to-machine (M2M) communication

One of the sub-topics in the IOT is machine-to-machine communication. And this area is already been actively explored by both enthusiasts and commercially by the industry

and businesses. The topic is very broad, but again, I mention it in the context of my research only. Briefly, I got the first idea of the integration system architecture during the study over M2M. I liked the concept of machines describing their assets to other machines, and enabling mutual collaboration on achieving common goals. I thought that it's great that M2M is getting more and more popular, because soon it will help users to integrate their devices in their interactive environments, like homes, offices, and public venues. But as of the present day, I am not satisfied with available M2M solutions. Although current systems help to solve lot of problems, they just don't seem to be solving mine. I am not yet in the stage where I care for the power efficiency management in my home, nor do I find it fancy to deploy m2m system to help me turning lights off when I forget to do so. With the current pricing the later would rather cause me to loose money than earn on return of investment. I am not saying these are not practical use cases, I am just saying that these are not my immediate needs. Instead, I would like to have m2m system, that doesn't require me to purchase expensive hardware, nor to have a service officer to help me to deploy it. A system, that would allow me to integrate devices in my house to my liking. I want a system that is lightweight, elastic, an in open architecture, so I can build my custom solutions on top of it myself. And I believe that there are more people like me. Yet, I am not able to find a suitable solution to address my needs. So instead, I eventually came up with my own, and I want to propose it to others as well.

1.6 Modern System Integration (SI)

After I had a clear idea on what I want to do, I needed to start from researching the prior art. I have studied into few areas that cover the problem of integration on different levels and for different types of components.

1.6.1 Home automation

The Smart Houses get quite a lot of buzz in the Internet these days, and living in Hsinchu where the Science Park is located allowed me to see that it's not only the thing that people discuss about. Lot of Tech companies are already providing solutions in that domain, or working on such solutions. What I have also observed is that there is much more focus on the Smart House appliances (hardware) than on Smart House frameworks (software). At least based on my observation of what solutions Taiwanese manufacturers are presenting to the market. And Taiwanese manufacturers are known to be powering a lot of worldwide technological advancement. I thought that it can be a good idea to address the market niche from the software side, and if implemented, perhaps propose it to hardware makers, who could push it to world's markets if they liked it. Quite a big challenge, but why not?

1.6.2 Sensor networks

While working with one of research teams lead by my thesis adviser, I moved my study from the home-scale down to on-body network of sensors. Naturally, the sensor networks do not have to be biosensors wore on user's body. They can also be part of home automation systems. But there are some very interesting problems specific to Body Area Networks (BAN). One of the problems I was careful to research on, was the need of real-time system responsiveness for certain applications. Another is the power efficiency problem, which in case of BAN, is even more crucial than in home automation systems. While in the Smart Home inefficient power consumption may at worst result in high electricity bill, in the case

of BAN if a sensor stops responding too early, it may have bad outcome on a patient. And while in home devices can be powered from the grid, in the BAN we would like to have an autonomic power supply (that we can wear and be mobile with it). I consider these challenges very interesting and worth addressing, but for the purpose of my own system proposal, I eventually understood that I want to limit the target for it. I understood that I want to develop a solution for the domains where RTT and low power profiling are not crucial. It does not mean that my design will ignore power consumption optimization, contrarily – it will allow for implementing power saving profiles. However, nowhere in the specification I will claim, that the system optimized for special conditions like in the case of BAN. Even though, I am sure that it can bring fair amount of limited value to specialized domains of use.

1.6.3 High level development tools

For all the less then recent domains of various system integration problems, there are plenty of tools and frameworks worth study. Throughout my study I found out that there are lots of tools that significantly speed up the work on integration tasks. And it seems (from my own observation and experience) that the higher level the tools are, the greater improvement in the speed of development. In fact, I realized that nowadays, we often address low level problems indirectly, via high level simulations and emulations. For example, emulating ARM architecture under x86 or x64 and developing under it, may turn out to be faster than actually connecting to the device and managing the cross-compiling and cross-debugging this way. But using Virtualization as

an example is not accidental. In fact, virtualization allowed me to see an interesting feature of high-level development. It allows for hiding things from the user experience. Running web service from within a VM will make no difference to the network. On the top level it will be exactly same as if the web service ran on an actual machine. Have you ever tried to run VM inside of a VM? This is an example of one high-level solution that allows for implementing other solutions on yet higher level in relation to itself. And finally it hides the fact that the level has increased from the user. I became particularly interested in this concept, as it allows for extending current designs by building around them, with the same effect as if the changes were made internally. I wanted my system to have this feature. If there is a physical device connected to the network, I would like to write a virtual one, that implements additional features, and then handle it over to the network just like if it was a physically upgraded device. Actually, the WHOLE system follows that psychology. It combines all the physical and virtual devices into ONE large virtual device. And if needed, it can then be added as a resource under another network (just like the VM inside of a VM).

2 Problem analysis

2.1 Problem definition

Although in current times we have access to a massive market for Customer Electronics and we have lot of standards of communication both wired and wireless, but it is not trivial for an average user to integrate his devices and programs into a manageable collective population of software and hardware resources.

Let’s think of a bare bone PC for a moment. Because it is equipped with standard I\O ports, lot of additional hardware can be attached to it, increasing its functionality. Connect a printer, and PC can print on paper. Connect a scanner, and PC can digitalize paper documents, even use OCR software to turn it indo editable digital text. Add WiFi network card and the PC can surf the Internet and connect to even more hardware peripherals wirelessly. It can now control your DSLR if it has WiFi support too. Add an IP camera and your PC can carry out surveillance in the house. It can communicate with a Smartphone. Connect an Ethernet-controlled power outlet, and you can power on and off just any electrical device around the house. And since the power outlet is connected to the PC, and the PC is connected to the Internet, and so is a Smartphone, an electrical device in the house can be turned on and off remotely from any location on Earth provided only that there is Internet connection available.

The way it has just been presented makes it seem that it is quite easy to carry out integration of devices into highly custom systems. This thesis explores the integration problem and later on introduces an alternative way address this problem.

Consider a scenario:

A fictitious user named Bill owns home audio system, PC, DSLR, and couple of mobile phones. Bill is not an engineer, but he’s a computer hobbyist. He would like to make his home more intelligent and interactive. He wants that when comes back home, the home can recognize him and play music based on his preferences. However, if Bill’s wife comes home first, the home would play music based on her preferences instead. And besides, every time there is a friend visiting, while Bill’s not home, the home would try to recognize friend’s face and look it up the local database. Voice message would greet the guest. Also, the home would email Bill with the photo of visiting friend and the time record on when the visit took place. Bill thinks that he can build such a system himself

using components that are aleary in his house, but were so far being used individually. He wants his DSLR to hang on the wall in front of the entrance and take photos in a time loop, then send them wirelessly to the PC. PC would run a face recognition software to detect people's faces on these photos (if any). If a face is detected the software would look it up in a database. If it matches Bill or Bill's wife, PC would play music from assigned playlist on the home audio system. Otherwise, if it is someone else, while Bill's out of home, an email with the person's photo would be sent to Bill. So if Bill's mom came by to feed Bill's cat while Bill's out of town, he would get notified by email. Or if a burglar has entered his house, Bill would have his photo to show to the Police.

In the scenario above Bill has components to address particular tasks within the scope of the overall problem individually. However, he needs to implement the integration to turn these "indivituals" into "team players". Turns out, that this is not a trivial problem to solve due to few key factors:

Selfish hardware	hardware that is designed to be able to connect to a network is not designed to serve as a resource to the network (a DSLR may have WiFi connectivity built in, but its design limits the way it can be used once connected to only a dedicated domain,)
Selfish software	Commercial software owned by the user is not designed to allow the use of its functionality outside of the dedicated domain either (a photo browser can have face recognition functionality, but it does not provide this functionality to other applications)
Middleware challenge	The difficulty level of implementing a middleware to enable software and hardware components to be used as shared resources arises as each component requires custom approach to building a standardized

	wrapper/handler/driver around it with custom communication protocol (be it IPC, Serial port or UDP or TCP/IP signaling, web API or other)
RPC challenge	A custom RPC-enabling layer would need to be developed prior to being able to implement high level code

The above table will serve as a break-down of the problem definition, and will get referenced to later on.

2.2 Current designs and solutions potentially useful to address the problem

Various protocols and standards have emerged over the past several decades to address the challenges of distributed computing. From EDI (Electronic Data Interchange) to SOAP (Simple Object Access Protocol), to Web Services with WSDL, there are quite some of standards allowing for integration and interoperability of various systems. These are on protocol design level and software level. On the other hand, for an inter-device integration the physical architecture also has to allow for communication to be carried out. In wired communication we have USB and Ethernet cable support as two very mainstream options for interconnecting various electronic devices. For the Wireless we have Wi-Fi and Bluetooth as two other globally popular standards.

Naturally, these are not the only ones, but I chose to focus on these as most common ones for connecting peripherals (based on my own observation on CE market and computer accessories/peripherals market). Pointing out these few standards will become especially relevant when discussing integration in the mainstream market, where end users care for easiness of connecting their devices. In the industrial environment, or in very specialized cases, some other standards might be chosen for matching certain needs more adequately.

There are several existing ways to address the factors of “selfish hardware” and “selfish software”. In practice, addressing hardware customization is in deed same as addressing software customization, because what we want to actually customize is the firmware within a device. Otherwise, we can customize the device driver, in which case customization takes place outside of the device. Yet higher level of customization is in the layer of the user application controlling the hardware through its driver, in which case the distinction of hardware and software customizations is no longer needed. From the user’s perspective there is no hardware versus software concern. What user cares about is the customization of the way a resource can be used. Addressing this part of the problem is possible through a range of existing technologies: from plugin-based development (ie. control-over-http plugins), to a much higher level M2M protocols, to solutions compliant with the Internet of Things. The IOT however should be considered a still-emerging solution rather than an existing one.

The middleware challenge (Ch. 2.1) has to be addressed by implementing a wrapper-code around existing drivers and/or user applications. During the research there were no 3rd party solutions found that can automate this process for a typical binary executables, but few wrapper-building frameworks were found for dedicated use:

1. Launch4j (http://launch4j.sourceforge.net) is a cross-platform tool wrapping Java applications distributed as jars in Windows native executables. The tool allows an automated building of wrappers because the JAR applications incorporate meta data describing entire headers of functions compiled into an application. In case of OS native binary executables, such meta data is not present, or is not complete. Shared libraries have meta data describing names of exported functions, that can be extracted, but not the list of the arguments. This has brought a conclusion, that there might not be tools to automate the process of solving middleware challenge (Ch.2.1) when past

the source-code stage. On the source-code level, there are many libraries that can add RPC support to an application, like XML-RPC or JSON-RPC to name just few popular. It will get examined later on, how much work from a programmer is needed to add RPC to an existing application. It will also be examined if the proposed alternative solution can actually turn out simpler.

2. W4F is an approach to build web applications based on middleware architecture with Web wrappers, also called adapters (Sahuguet et al, 2001). This case is worth mentioning because it shows that not only a de-facto application can wrapped. W4F communicates with data sources published in the Internet, not applications. This brought the final conclusion: anything that contains enough of meta description of itself, can be integrated with a middleware layers relatively easily and allowing automation in the process – which is not the case with OS native binary executables.

When it goes to the RPC challenge (Ch. 2.2), it is passed the concern for how to integrate particular RPC standard with existing application. In this stage the concern is actually deploying the RPC environment. In a generic scenario for RPC standards mentioned before, there is client-server environment to be built. Chapter 3 will propose some alternative in details of such architecture.

A new solution for integration and interoperability of SW and HW resources would have to somehow bring improvement as an alternative to use technologies mentioned above. Otherwise it would need to allow for incorporating these technologies while still introducing significant improvement to the overall problem.

2.3 Defining targets for potential solution

2.3.1 Value to users

End users are the principal target because the main focus of this research is put on the ability of end-users to integrate their devices with their

computers and applications running on them. This target group will naturally care for an integration solution to be inexpensive, easy to deploy and use, with user-friendly interface, and to be very popular, so that there is a big community working on it. This target group will also care less for some very advanced and low-level aspects of the architecture used for realizing the integration, as long as the solutions brings them positive user experience. One exception is the energy consumption efficiency. It is where users will likely care to a greater detail. Users can find it generally attractive to have their homes delivering the Sci-Fi feel of being able to control all the devices as a collective network that they can program to reach high level goals. That kind of experience would follow the concept of a smart home, which is a growing trend:

"*The global home automation systems market has been forecast to increase at a compound annual growth rate (CAGR) of 35.5% through to 2015, having already boomed from 237,000 systems shipped in 2007, reach more than 4 million by 2013*" (King, 2012).

And the home automations, apart of the user experience aspect, is the key to having effective energy management in the house:

"*Systems from Savant, Crestron, Vantage and others can even be programmed to turn off or turn down devices if preset energy use ceilings are reached—though this will likely require programming costs. Even lower-end systems being sold by the likes of ADT, Comcast, Verizon, Vivint and Alarm.com can be programmed with basic macros and timed events to turn things off when someone leaves the house and the security system is armed.*" (Castle, 2011).

The proposed solution can address the power efficiency concern in at least two ways:

1. its own architecture can follow a design optimized for minimalizing power usage,
2. it can provide an interface for 3rd party systems to deploy power-saving policies by issuing commands to power devices on/off or to put them in (or wake from) the sleep mode.

2.3.2 Value to businesses

The value to the business is dual: one set of benefits is related to the internal use (powering the micro environment = cost) and another relates to addidional business opportunity from having end users as primary target for the solution (powering the macro environment = revenue). Firstly, the solution may help the business to run its operations in more elastic, perhaps easier, less expensive way as compared to hiring System Integrators for implementing highly customized solutions. Saving costs is also earning.

Secondly, any popular standard or feature is potentially a direction for a product or service providers to create an added-value to end users as there will be demand for it. Ideally the solution should give businesses a competitive edge in either introducing innovative features to their products, or just simply allowing them to implement more products and services (or same number but faster or cheaper) than without the solution. And with more compatible/compliant products on the market the more growth of the solution's market share is expected.

2.3.3 Value to industry

The proposed solution is not especially targeted at industrial use but is dedicated for spreading through manufacturing industry.

When looking at how brands are selling their CE products it is easy to see that the hardware factor is the least crucial in determining which brand will take a market share away off another. Essentially, vast majority of branded products is being built by OEM manufacturers and in terms of specs the whole world's market has access to pretty much same resources, with only difference in costs, as every buyer gets his own deal with the supplier. What end users see as differences in specs, is actually a result of strategic product segmentation, and has little to do with technical advancement of the brand itself. Having essentially the whole market getting same hardware technology from same primary sources (major OEMs), what gives their products competitive edge is largely the design and features. The more interesting the design and the more feature-packed the product is, the bigger chance it stands to capture end-user's attention. And there is an aspect here worth more study: Customers recognize brands and pay high margins for big names, while OEM manufacturers who remain largely unknown to end-users, and who sell products at low costs minimally above the cost of production (but in large volumes) are the ones who makes the most money at the end (like when realizing that Apple considered by end-users as a brand making lots of money falls short compared to the money that their supplier Foxconn make*). How is that relevant to the research covered here? A solution proposed later on in the chapter 3 can potentially have a significant influence on the manufacturing industry, from OEM to brand-level. When a user buys a

Smartphone, the brand, the design, the features are all important. A device like Smartphone is a self-contained product, and via implemented wired (USB) and wireless (Wi-Fi/BT) standards, can communicate with lots of other devices. But built-in specs are not designed to be exchangeable/upgradeable like in PC computers. When it comes to DIY approach, and end-users interested in adding features to their intelligent and automated environments, things become more interesting. A cheap, "brandless", simple sensor, that would be no value to an end-user if it was not self-contained and able to be used stand-alone, becomes a tasty treat if the same user can just bring it home and add to an existing network of resources to enable new capabilities to his automated home. Actually, one of more important reasons why end users would not buy very cheap and common electronic components, sensors etc. and would instead buy more expensive, more complex, self-contained, branded products, is that they would not know how to make use of these cheap components anyways. It's simply easier to buy the later, even if one ends up with a product packed with sensors and other built-in components, only to find out that lot of them will be at no use to him. This can change dramatically if there is a common integration solution, that would allow anyone to buy single, inexpensive components, bring them home, connect and just use. It would change a lot in the market, if in order to have our homes to intelligently response to environmental conditions we would only need to bring a box of cheap sensors from an electronics shop and have it all being used in conjunction with the PC, laptop or a Smartphone that is already there and has all the computing power needed to run it all as one complex

distributed device. Currently, although far more expensive, it is just easier to either buy a dedicated system including a hardware + software bundles and have it deployed in our house, or to hire a System Integrator.

In the same time, it would not necessarily mean that OEM manufacturers would start to go directly to end-users causing brands to loose business. It would give businesses more opportunities to sell more, with less effort, at lower prices and with lower cost. The reasoning behind it is, to put it the simplest way, is that most of end-users are lazy. Even if an end-user can easily DIY his own automated home, he will probably (author's point of view) not be very interested in developing all the applications to be run in his integrated system by himself. This is where business can start making more business. Buy cheap components, spend no time on external design, and branding, but implement features, and write applications that will be able to co-use these features with other features that will be available at the run-time when the product becomes a member of a larger distributed system. There are at least 2 possible changes that can happen in the manufacturing industry if the proposed solution went viral:

1. End users would expect devices to be increasingly compliant with the new standard, so that after purchasing goods there is more freedom to create applications for these goods even outside of the domains they (devices) were dedicated to serve originally
2. Manufacturers would be able to start targeting mainstream market with more component-like products, which have little market value as stand-alones.

2.3.4 Value to developers

Not only hardware makers can potentially find new ways to design and sell products, but also software developers can be given similar opportunity. Looking at any end-user level software there is a large number of functions implemented in it, but in general these functions are only being used to solve problems that the program containing it was dedicated to solve. What if an end-user figured out another way to use certain features of a complex program to solve a problem that was not expected by the program writers to be solved with it, or was not thought to solve the problem in the same way? Unfortunately for the user, there might not be a way to extract a subset of functions from a program and call them from outside of that program (unless the software was originally meant to be used in this way). Perhaps allowing the user to be able to selectively call functions of a program that he finds useful could make a great added-value to the software product. It could also give software sellers more possibilities of implementing sales. For example there could be a price structure for a software based on a license that either allows the user to selectively call its sub-functions remotely or not. Perhaps it could give more flexibility to sell SaaS (Software as a Service) where user can selectively call functions based on his subscription. Google has released a freeware software called Picasa, that allows to manage, browse, edit and share photos. Among many features contained by it, there is one that allows for face detection and recognition. It is a great functionality, but Picasa only uses it to tag people on photos and allow search in the library of photos by presence of people tagged on them. If that functionality could be shared out, then a photo taken by another device could be

analyzed and if a face was detected and recognized some further action could be taken based on the identity of a person or persons (and that further action could call functions from other programs or other devices). Picasa is only one of many software products offering face recognition and tagging functionality. Very similar features will be found in Adobe Photoshop or Corel Paintshop Pro. There is even a chance that the software maker has not implemented its own computer vision library, but that he licensed a third-party library. It is even possible that same library is being licensed by many software makers. Perhaps that original library could be added as a resource to an integrated system, so not only software makers, but anyone could use it (be it for free or by purchasing a license). And then Google, Adobe and Corel could release their software without the built-in face recognition functions, but marketing their products as compliant to a general standard, so that they can integrate with computer vision library if only one is present in the system where the software is running. Just like a Mobile Phone Carrier can buy a Smartphone from a vendor, put a logo on it, create a custom ROM with Skype application installed, and sell it advertised as a phone with Internet Call support. Even more interestingly, instead of having many programs running on a machine having their own "copy" of the vision library loaded into memory, there could be only one instance of the vision library and all the programs on all the devices connected to the system could use it as a service, or as a shared feature.

The integration system itself would be a new type of platform for developers to write applications for. A platform, in which, features are

distributed across devices and programs. Where any feature can be either software or hardware implemented, while it remains transparent to the user. Where applications could be designed to use resources before knowing what these resources might be during the run-time. And possibly in a programming languages that handles RPC transparently so an RPC looks like a regular local call from the source code level. This will get further discussed in chapter 3.

2.4 Designing benchmarking criteria

2.4.1 Objectives

The principal objective for proposed solution (see Ch.3) is to improve (or replace) alternative/competitive solutions on the market. Even if the competing systems incorporate more numerous and/or more advanced features, there are still other factors that might potentially help in reaching better reception from the market: simplicity to end-users, simplicity to developers and price effectiveness factor. The more complex (feature rich) a product is, the more difficult it is to keep a simplistic, intuitive interface for end users. There is always a chance, that the user when comparing products, can choose the one with a smaller feature set because of a simpler, more intuitive interface and a better user-experience. Since the evaluation of these objectives will not be possible without actually releasing the solution to the market and waiting for market's feedback, a benchmarking model will be designed to approximate system's potential to compete with other solutions with more measurable criteria.

2.4.2 Factor 1: Simplicity to End-users

This factor is related to how well the proposed solution can satisfy End-users liking in aspects such as:

- Intuitive and simple User Interface (UI)
- Low learning curve needed before able to deploy and use
- Seamless scalability: growing the system require the little work and special attention
- Richness of applications: less advanced system stands chance to become more popular against a much more advanced competitor if it exposes significantly more use applications for the end-users (becomes useful for more things)

2.4.3 Factor 2: Simplicity to developers

This factor is related to how easy it is for developers to build more solutions on top of the provided system, or to integrate it with other systems. The easier it is, the more development projects are likely to grow around the proposed system, and use applications go to End-users (helping the first factor).

Another aspect is that once developing under the proposed system becomes simple and innexpensive, it can help more End-users to also become developers. This can grow the rate of End-user development (EUD) for the system. Computer users nowadays stand for a huge number and diversity (Scaffidi et al 2005), spreading across many disciplines and including professionals in many fields: managers, accountants, scientists, engineers, teachers, health care workers, administrative workers, even home makers and many more. Their work related tasks may dynamically

change on any basis from annually to even daily. If so, then also their needs and expectations for any software used vary frequently, and become more complex. Professional software makers fall short in meeting all these needs directly due to their limited domain knowledge or/and because of too slow development process (Burnett and Scaffidi 2011). This shows how increasingly important is the scale and role of EUD. One of the definitions of EUD states that it is “a set of methods, techniques and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify, or extend a software artifact” (Lieberman et al 2006). End-users can particularly design and/or customize the UI and functionality of a system, when they need to better fit their specific context and needs. Also, because end users outnumber professional software developers by a factor 30:1 (Burnett and Scaffidi 2011), EUD enables much more people to participate in development. In order to encourage more EUD happening, some specifically appropriate tools and features are significantly important, that can make development processes highly usable and quickly learned. It is because unlike professional developers focused on durable software assets producing continuous revenue streams, the end users rather do programming for reaching short or medium-term goals – sometimes they just want to write code “on the spot” to use it instantly, and without carrying if they will later need to ever use it again. Enabling more EUD over the system requires a proper, dedicated development framework. Because the discussed system is dedicated to enable integration and interoperability of hardware and software resources

forming a network, it shows lot of similarity to End-user programming of Smart Homes. This is a very challenging issue. A great way to realize scale of complexity was to propose a model for networked artifacts inspired from molecular chemistry (Coutaz, 2008). To provide end-users with the capacity to "program" their interactive spaces like their home, a way to handle complexity of networked artifacts must be found. There is an analogy to this complexity found in chemistry where a smart artifact is modeled as a composition of physical and digital atoms whose configuration evolves under particular conditions. Software services are composed of digital atoms only, whereas conventional objects of the real world are strictly built from physical atoms. A smart house is considered as an interactive space represented by a unique macro-molecule of both digital and physical atoms, or a set of smaller molecules. Then just as in Chemistry, the nature of the events that trigger a reaction has an impact on the resulting product. This way to look at the EUD problem would rather discourage average end-users to participate in the development, which is why hiding all the complexity from the user while still allowing him to produce results, becomes crucial.

2.4.4 Factor 3: Price effectiveness

New system should allow for an inexpensive integration. An average user should be able to afford using it. This factor is much more reflecting the business aspect then just focusing on providing technology solution. While the actual marketing planning is not within the scope of this paper, it is relevant to some extent when it comes to architectural design. Regardless of the form in which the sales of the solution might be

implemented in the future, in the architecture level it should be considered how to accommodate as many possibilities as there might be. From the Chapter 1 we have seen some new trends in sales design surfacing, like selling software or infrastructures as a service for example. Even though this paper only acknowledges the business aspect to become relevant at some point in time beyond the scope covered, but keeping it in mind can greatly help in designing system architecture.

2.4.5 Justifying the selection of factors

To justify the 3 factors given a case study of Android's (operating system) market share was arbitrary chosen. Below are the worldwide smartphone unit sales and market share in the 1st quarter of 2012, by IDC (FRAMINGHAM, Mass. May 24):

Mobile OS	1Q12 Unit Shipments	1Q12 Market Share
Android	89.9 mln	59.0%
iOS	35.1 mln	23.0%
Symbian	10.4 mln	6.8%
BlackBerry OS	9.7 mln	6.4%
Linux	3.5 mln	2.3%
Windows Phone 7/ Windows Mobile	3.3 mln	2.2%
Other	0.4 mln	0.3%

Furthermore, from the research by IDC we can see growth of the above market shares compared to Q1 2011:

- Android – 145% growth

- iOS (iPhone) – 88.7% growth
- Symbian – 60.6% decline
- BlackBerry – 29.7% decline
- Linux – 9.4% growth
- Windows Phone/Mobile – 26.9% growth

Particularly interesting is the growth of Android and the second biggest player on the market, the iOS. How an OS can reach such success in a crowded market? Firstly, let's focus away from the fact, that part of Android's success has come from Samsung, which accounted for 45% of all Android sales in the quarter. Still, Android is the latest into the market from all the top popular OSes in the Mobile market:

"The popularity of Android and iOS stems from a combination of factors that the competition has struggled to keep up with, (...) Neither Android nor iOS were the first to market with some of these features, but the way they made the smartphone experience intuitive and seamless has quickly earned a massive following" (Ramon Llamas, Research Analyst with IDC).

That illustrates how a newly proposed solution can win the popularity among end-users if only focuses more than the competition on user's experience. Another crucial aspect is the number of available applications, in which case, using the mobile OS market as a case study brings in even more interesting observations. From the top level, Android and iOS both are leaders in the number of apps available to their users. This measurement is greatly influencing the end-users when it comes to deciding whether or not to start using particular product – in this case it is

an operating system, or in broader context, a Smartphone powered by a particular OS. This case study will make few key observations in process, that will reflect in the thinking on the solution proposed in Chapter 3. To start with, let's point out, that in the case of mobile OS market, the software becomes a great deal when it comes to the value of a hardware. One could speculate if a Smartphone great in hardware design, could represent a high value to end users if it was running on an OS with very low number of applications. This question also reflects consideration for the solution proposed in this paper, that is also implemented in software. But there is more to learn from the example of Android and iOS study in the area of apps. Great observations has been made by Darcy Travlos in "Five Reasons Why Google Android versus Apple iOS Market Share Numbers Don't Matter" for Forbes. She has made some research on Gartner's Q2 2012 mobile sales unit report, and some of her findings are as follows:

- Apple does money on hardware, Google does not
- Google gives away its Android operating system and earns on Search and delivering ads
- Apple makes money on every iPhone and iPad it sells, even before an ad is delivered to the device
- Apps are more popular than mobile web, and Apple is winning this race by any measure
- on mobile devices, 94 minutes per day are spent on apps compared to 72 minutes on the web
- Apple offers more apps, and more are downloaded from Apple

- people are spending for Apple's apps – Apple's apps make money
- App store generates $5.4M/day for the 200 top-grossing apps while Google generates just $679K for their top-200 grossing apps. That is almost a 8:1 revenue ratio
- More of Apple's apps generate revenue, while most of Google apps are free (67% of apps on Apple are paid for versus 34% on Google)
- Apple apps make developers money (Android developers made $210M in all of 2011, compared to the $700M pocketed by Apple iOS developers in the Q4 2011)
- getting paid attracts more developers to Apple
- ppStoreHQ estimates there are over 43K Apple iOS developers and 10K Android developers (Travlos points that it is because iOS developers earn more)
- By estimate, for the very same app, a developer will earn $1.00 on the Apple iOS version compared to $0.24 for the Google Android version

After looking at the findings listed above, a very important consideration for the research comes to mind. Even if the focus of this very paper is on proposing a technology-based solution, not a business plan, the aspect of a potential revenue model for the solution if delivered to market, seems clearly crutial to satisfy objectives presented earlier. The 3rd factor (Simplicity to develepers) in particular requires the system architecture to consider later potential earning opportunities for developers. Before concluding this portion, it is further helpful to consider 3 reasons for why many new popular apps are developed for Apple iOS first, given by Travlos in her article:

- Apple has fewer form factors (3 iPhones, 3 iPads) compared to thousands of Android devices
- Apple's app approval process requires developers to guarantee a certain quality
- Apple user demographic is more affluent, an earlier adopter and more loyal than other brands

What constructive does the fact that iOS developers have fewer sets of hardware and middleware issues to address than do Android developers bring to the research covered in this paper? A suggested architecture will be targeted to numerous and diverse base of hardware platforms. In order to achieve Apple's advantage, the developers under this architecture should be given a very generic development environment, that can hide as many form-factor-specific details as possible. At least in greater deal than the competitors have managed to. Another Apple's advantage is in building credibility in guarantying quality apps (apps approval process). In this paper the research does not stretch further into the app market business model, nor does the proposed architecture include it, but it should propose alternative ways to achieve credibility among users (considering specifically the use applications). As for Apple's third advantage, the scope of this paper does not reach to the problem of user's loyalty to a brand, but some particular demographics can be especially targeted, to reach as many of the early adapters as possible once the system gets released.

The third factor treats about the appealing pricing strategy. To create an advantage in the system design, the architecture should support the

integration with various possible implementations of sales strategies. In particular an effort should be made to design the system in the way to easily enable some of the trending sales approaches.

2.4.6 Benchmark summary

To allow comparison with other existing solutions, a benchmarking model will consist of two units: User Experience Unit, and SDK Unit:

User Experience Unit for benchmarking	
Intuitive UI	Does the solution offer a dedicated Graphical Interface for end-users that is easy and intuitive?
Learning Curve	Is the learning curve reasonably low for end-user to deploy and use the system?
Scalability	Is it made easy for end-user to increase number of resources in the system?
Applications base	Is there a public base of applications for end-users to use?
Affordability	Is it expensive for an end-user to get the system, deploy it and use?
SDK Unit for benchmarking	
Package	What does the SDK package consist of?
Learning Curve	How much new knowledge a developer needs to acquire before being able to develop for the system?
Portability	Is it easy to port existing project to this new framework?
EUD support	Is there a natively built-in development kit for End User Programming?

3 Solution proposal

3.1 Overview

Designed is a middleware system for allowing the integration of arbitrary electronic devices and software into a collective population of resources capable of cooperation. The complete architecture consists of 5 layers, that are covered in

subsequent chapters.

3.2 Layer 1: Distributed API

3.2.1 Components of the system

There are two types of components that can exist in the system: API Operator and API Contributor. The system includes at least one API Operator, and nay number of API Contributors. API Operator is a server application that collects RPC calls from connected API Contributors (clients) and presents it to end user via graphical interface (GUI). The GUI allows for listing and invoking distributed RPCs. For a network there is only one API Operator, and it is the End-user's only interface to invoke RPC throughout connected API Contributors. However, the system can be configured to allow one API Operator component to connect to another and create a sub-network (analogy to network routers). It is because the API Operator also can open client connections to other servers, in which case it will forward all its API listing to another API Operator above it. Shall an end-user connect to an API Contributor that is connected to another, will be informed about the existence of a higher level API Operator and will be given a link to it, while he will still be served normally.

Every API Contributor has a built-in domain ID (DID). When API Contributor connects to API Operator, it presents itself with his built-in DID, and listing of its API, that may look like this:

```
DID: AirConditioner
-- API listing: ------
Procedure PowerOn();
Procedure PowerOff(delay: integer);
```

```
Procedure SetTemperatureCelcius(temp: integer);
```

After that, the API Operator will list its API as available to the End user, under the domain “AirConditioner”:

```
-- API listing for AirConditioner: ------
Procedure AirConditioner.PowerOn();
Procedure AirConditioner.PowerOff(delay: integer);
Procedure AirConditioner.SetTemperatureCelcius(temp:
integer);
```

And the end user will be able to issue its API by sending RPC calls to the API Operator in the format:

```
PowerOn@AirConditioner();
SetTemperatureCelcius@AirConditioner(25);
PowerOff@AirConditioner(30);
```

If the API Operator is also connected under another, than the above calls can also be passed to that another one. In such a case, the top level API Operator will assign a sub domain to access lower level API. Assuming the given sub domain is “subnet1” the calls will be published as:

```
PowerOn@subnet1.AirConditioner();
SetTemperatureCelcius@subnet1.AirConditioner(25);
PowerOff@subnet1.AirConditioner(30);
```

More details are covered in the subsequent chapters.

3.2.2 API Operator

The API Operator is distributed as software for downloading and installing in

the end user's LAN. It can be downloaded as installer package for a dedicated platform. An API Operator has at least two server sockets: one for API Contributors to connect (port #8201) and one for HTTP on port #7777 (assigned as a standard for the system) for the end user to access its GUI. However, it can incorporate more server connections for providing more access methods for end users or for other custom needs.

The API Operator is required to be given more computing power than API Contributor components. The system architecture is designed in the way, so that the API Operator takes all the load of processing and computing on itself, while API Contributor only receives control commands and responds. The API Operator has a built-in interpreter that will parse RPC calls, and check the syntax, do the type control etc. API Operator also has a built-in memory management system for dynamically creating memory structures like variables and objects. Its interpreter supports a simple built-in scripting language for End User Programming. The API Operator is dedicated to be installed on a PC or mobile computer (PDA, Smartphone, tablet, etc.) where enough computing power and memory resources are expected to be provided. It takes the high level syntax from the End user's input, checks if against errors, interprets and converts to simplified, low level control commands to be issued to API Contributor components.

3.2.3 API Contributor

The API Contributor is available for download as shared libraries (ie. DLL in case of Windows family operating system) and can be added to an existing code for a software or embedded system, to make it compatible with the system. Compatible software and hardware, after connecting to end user's

LAN, will be able to automatically detect API Operator component (if present) and be configured by it. API Contributor has minimal requirements for CPU power and memory. It does depend on the API Operator to carry out the business logics. It simply responds for simple control commands with no additional work. This way allows the whole system to have one dedicated hardware for API Operator, that will use the power for CPU, and that will use memory, while other devices with API Contributor built in can function in power-saving manner. Moreover, the default way for API Contributor to connect to API Operator is by pooling, in the connectionless manner. This can further increase power saving. API Contributor, while pooling, will download a list of control commands, disconnect, and execute the commands. The next time it connects to the API Operator, it will send out the status messages and fetch new list of commands.

Another feature of the system, is that in order for a hardware device to be compatible with the system, a corresponding API Contributor component is not required to be embedded in the device itself. For example, a device can be connected to a PC over USB cable, and use a custom driver. In the same PC, an application can be installed that can be an agent between the device and the system. A different example may include a sensor that has built-in support for low-power Bluetooth communication. The API Contributor can run as an independent agent application on a PC or on a mobile device (ie. Smartphone) and represent the sensor in the system. API Contributor can be implemented as a Protocol Adapter (Protocol Gateway) to represent a resource supporting virtually any standard of communication.

3.2.4 Buffered RPC protocol

As the API Contributors use pooling to connect to API Operator, the control commands that need to be handled are first buffered by the API Operator. When the API Contributor connects, it will be handled the list of control commands, and will be expected to return a status of execution in form of simple messages on the next connection. The buffering happens transparently to the end user. The API Operator will flag individual resources as available or unavailable (online or offline) based on status messages (including responses to PING signals) and timeouts.

3.2.5 Self-hosted API Operator

For some cases a device or software maker may want to use the system exclusively for it's own commercial application. This is possible via so called Self-hosted API Operator SHAPIO). A device or a software would include a SHAPIO library in the existing code, and set a custom port for an end user to access its GUI. By doing so, the final product is added a custom RPC interface, including EUD scripting language support. The advantage of such approach is that the RPC support is added to the product with entire Front-End and it is fully independent (as opposite to the default architecture where a central API Operator hosts the network). A disadvantage is, that by including the entire implementation of API Operator (Memory management, parser, communication, etc.) the products need for CPU power and memory consumption increases. Whether the presented advantage is disadvantage has a stronger relevance is a matter of a particular application.

3.2.6 Virtual API Contributor

A virtual API Contributor is a component of the system, that shares an API

without built-in facilities to perform related actions. Instead, it “hires” other components of the system to perform actions. A use of a virtual PI Contributor can be useful to:

- create agents (enable in-compatible resources)
- integrate multiple existing components into one dedicated
- adding an API Operator to another network as a resource

As an example, imagine two components with built-in API:

```
DID: EPO (Ethernet Power Outlet)
-- API listing: ------
Procedure PowerOn(SocketNumber: integer);
Procedure PowerOff(SocketNumber: integer);
DID: AMS (Ambient Light Sensor)
-- API listing: ------
Function LightAmount(): integer;
```

The first one is a power outlet and the second one is the ambient light sensor. The power outlet has 4 sockets that can be controlled remotely. There is a lamp connected into the 1st socket. The ambient light sensor measures the amount of ambient light and returns a value from 0 to 100 where 0 means no ambient light, and 100 means the amount of the light in the full daylight (or higher). Notice, that the lamp connected to the power outlet is not system compatible, and does not share any API. We can create a virtual lamp with the setup described above, with an API listing as follows:

```
DID: SmartLamp
-- API listing: ------
```

```
Procedure OperationCondition(AmbientLightLevel: integer);
```

The SmartLamp will be added as a virtual resource to the system. The network will not care, that it has no physical representation. Instead, it is a software, that pretends to be a smart lamp, and uses other resources to perform actions and deliver outcomes. It has one procedure in it's API, it allows for the end user to set a tolerance for the amount of the ambient light. When an amount lower then the specified is detected, the lamp turns on. The way the virtual API Contributor delivers the outcomes is that fetches the ambient light sensor checking the current amount of light, and if a condition is fulfilled (when the value goes below the set limit), it controls the socket on the Ethernet power outlet to which the lamp is connected. The physical switch on the lamp is permanently turned on, and by controlling the power outlet, the lamp can be either turned on or off.

A more powerful benefit of using virtual API Contributors can be illustrated by modifying the above example:

```
DID: SmartLamp
-- API listing: ------
Procedure PowerOutletDID(DID: string);
Procedure SocketIndex(index: integer);
Procedure OperationCondition(AmbientLightLevel: integer);
```

Now, the Smart Lamp is configurable and can be pointed at the end-user defined power outlet and a specific socket on it, to tell the smart lamp where exactly the physical lamp is connected to. Just in case there are more lamps, or more power outlets in the network. In the same manner, a custom ambient light

sensor can be pointed. Furthermore, a built-in PING call can be used by the Smart Lamp to determine whether or not a particular resource is available for delivering an outcome. And if the resource is not available, the end user can be notified. So if an End user Bill sets up the smart lamp for his home, but he does not have an ambient light sensor, he will get notified to get one. When he buys one in the local electronic shop, and brings it home, he can then set the smart lamp to use it. More complex API and implementations of virtual API Contributors may allow for scenarios like a smart lamp that can detect a physical resources after they were moved and connected to different power outlets, or can automatically recognize resources of certain brands, or can take consideration of more data that are specific to end-user's preferences stored in the configuration of his network.

3.2.7 API Contributor integration to existing code

The way that API Contributor component gets embedded into an existing code is by a shared library. The library contains fully-featured implementation of the component including network connection layer, pooling protocol layer, and RPC protocol layer. The developer, after including the library in his code, needs to export selected functions from his existing code through the API Contributor component to the network. This procedure only takes 2 simple steps:

1. Write a wrapper around a function to be exported
2. Pass the wrapper function as an argument to component's API registering function

Below is the fragment of source code listing showing the steps listed above, written in Delphi (a dialect of Pascal programming language):

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, TnTForms, StdCtrls, libEngine;
(...)
type
  TForm1 = class(TTnTForm)
    Label1: TLabel;
    Edit1: TEdit;
    Button1: TButton;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
    (...)
    Procedure VEcho(const FunctionName: widestring;
                    const Arguments: TVarTable;
                    const ArgCount: cardinal;
                    var FunctionResult: widestring;
                    const ResultType: widestring;
                    const RessultFlag: TTypeFlag);
  public
    { Public declarations }
    Function Echo(s: widestring): integer;
    (...)
  end;

var Form1      : TForm1;
implementation
{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
 libEngine.HostedFunctions.RegisterFunction
('function echo(s: string): integer;',VEcho);
end;
```

```
Procedure TForm1.VEcho(const FunctionName: widestring;
                       const Arguments: TVarTable;
                       const ArgCount: cardinal;
                       var FunctionResult: widestring;
                       const ResultType: widestring;
                       const RessultFlag: TTypeFlag);
begin
 FunctionResult:=IntToStr(Echo(Arguments[1]));
end;

Function  TForm1.Echo(s: WideString): integer;
begin
 Result:=Length(s); ShowMessage(s);
end;

(...)
End.
```

The code decorated with red bold font is the code that the developer needs to add to the existing code after previously including the API Contributor library. The **libEngine** in the list of units refers to provided library unit that includes and handles the shared library. The example above shows how an existing function **Echo** is exported. First a wrapper function is declared, and it is of a standard format:

`FunctionName` – contains the function name as expoted

`Arguments` – an array of string represented arguments passed to the function

`ArgCount` – the number of arguments passed to the function

`FunctionResult` – the string represented value for the result of the function

`ResultType` –type of the result that function is expected to return represented by a string ID

`ResultFlag` – same as above but represented by a type constant

The arguments of the function call are providing the developer with most important information that he might want to use in order to pass data to the built-in function. Let's first see how this wrapper is being exported:

```
libEngine.HostedFunctions.RegisterFunction
('function echo(s: string): integer;',VEcho);
```

The way a function wrapper is exported to the API Contributor component is by passing a function header in pascal syntax, and the wrapper itself as an argument. What will happen, is that the API Contributor will assign the VEcho wrapper to be executed only, when the API Operator matches the RPC call to the given syntax (that effort is taken off the developer's hands now). The system will make sure, that the RPC call is only passed to the developer's application if the syntax, type control, variable substitution etc. is properly executed beforehand. Only then, the function wrapper is called:

```
FunctionResult:=IntToStr(Echo(Arguments[1]));
```

Since at this point the information passed to the wrapper is verified and error-proof (by the API Operator), the End user can trust that there is a string argument in the first element of the `Arguments` array, and can pass it to the built-in `Echo` function. Also, the result of the function can be returned, but remembering that it has to be converted to string representation.

This example shows, that there is a very minimal effort from the developer side to enable functions from his existing code over RPC, when using the system. The content of a wrapper function will in most cases be only one line of code. It is a straightforward passing of arguments to the built-function with eventual conversion from a string to required type, and converting eventual result to a string.

3.2.8 Self-configuring feature

The system is able to configure itself automatically to a certain extent. When an API Contributor connects to a network for the first time, it will try to detect presence of the API operator either by checking default addresses or by using network broadcasting. Once the API Operator is found the API Contributor will present itself with a Domain ID and few other details like Serial Number and/or model number (optional). The API Contributor may provide the API Operator with a driver, or a driver might be delivered to the network from a separate file provided by the End user (or fetched from given, or public database). The driver is a plain text file containing Pascal syntax based listing of functions provided by the API Contributor. A single API Contributor may work under multiple networks, by putting multiple API Operator addresses on his list for pooling.

3.2.9 Custom GUI skins

Any resource of the system can be provided with a custom GUI. That is by preparing an HTML file, that contains designed UI and Javascript that will send RPC calls to the API Operator. This is to provide the end user with simpler interface than the custom support for scripting language. The custom GUI may allow for the end user to interact with a resource without writing code. Javascript will translate user's actions on the website into sequences of RPC calls, and then, it will translate returned status messages into visual outcomes. Any resource can be provided with multiple custom GUI skins, and one can always be configured as the default.

3.2.10 API Contributor drivers – the unique feature claim

This is a feature that I claim to be unique and which I believe is one of major

differences between the proposed architecture and other existing solutions. At least it is original when comparing to a competing solution discussed in the chapter 4.4.A driver is sim. ply a plain-text listing of RPC calls exported by the API Contributor. Without these listing, the API Operator would not now what RPC calls the API Contributor accepts. It is very similar to use of header files in C and C++ programming languages. The driver might be delivered to the system in two ways: either by the API Contributor itself or externally, from a network location (via download) or by uploading a file (web page embedded html form). When the API Contributor introduces itself to the API Operator in the network for the very first time, it is asked for the driver (or simply a listing of exported RPC calls). If the API Contributor is configured to do so, it will send the driver to the API Operator and the resource that its representing will get installed and will be listed for the entire network to share. Otherwise, the API Operator will first attempt to fetch the driver from a database specified in the network configuration, and if that fails, will create an alert to the End user, so that he can upload the necessary driver via the default user interface.

The role of the driver is much more significant than just listing available API. The system allows for multiple drivers assigned for each individual resource (API Contributor). Drivers my come in different editions, where each edition can selectively expose different subset of the API actually supported by the API Contributor. A different set of drivers may be distributed to different users, according on access policy. Also, the vendor/distributor of the resource, may implement different sales options for the exactly same product based on the driver version requested by the user. A free version of the product may come with a driver exposing limited API, while the full-featured API can only be

accessed when purchasing a complete driver edition. Also, different versions of drivers can be shipped for profiling the network. Imagine a scenario in which the system runs in an office, that has fixed working hours. During the normal working hours, the full API is available through the complete driver. But after the working hours, another profile of the network replaces the drivers, and only a limited API is available, disabling eventual users from accessing certain services.

3.3 Layer 2: Low-level EUD

This layer is an extension of the Layer's 1 built-in RPC protocol. The Layer 1 incorporates a parser (Lexical Analyzer + Syntax Analyzer) for interpreting RPC calls passed as unformatted text. Level 2 layer extends the parser and provides support for a scripting programming language dedicated to End User Development programming. The language syntax next to standard grammars like loops, conditional statements etc. incorporates dedicated grammars for handling RPC flow, with special consideration to:

- Availability/unavailability of the end-points for RPC calls
- RPC return statuses
- Connection errors during RPC transactions
- RPC expiration
- Blocking and non-blocking RPC

The RPC end-points are resources that RPC calls are being issued to be executed on. RPC return status is the final element of RPC transaction, after the end-point has executed the call, and when it confirms the execution to the API Operator and passes an optional result of the RPC call. If before sending an RPC call to the end-point and receiving the RPC status a connection breaks, there is no feedback

allowing to know whether the call has been successfully executed or not. Level 2 EUD programming language implements event-driven grammars to handle this kind of situations. RPC expiration is another built-in language feature allowing to optionally adding conditions on an RPC call defining an acceptable time before issuing the RPC call and receiving its execution status. In blocking mode, the code execution within a running script will wait for either the current RPC execution confirmation, RPC expiration or connection related event. In the non-blocking mode, the script will not be ran in the sequential way, but in a manner similar to a state machine, where an event will be an analogy to a state. On the syntax level, these events will be implementable as jumps, similar to how it is done in modern programming languages with a "goto" statement.

In this level the source code can be passed to the interpreter either via html Query String argument (or html form) or via a script file on the server, in which case the file will have very similar application as server-side scripts in PHP, ASP, Pearl etc.

In fact, a custom interface, which (as discussed earlier) is a static html page, can include script for interpretation, if this layer is implemented.

3.4 Layer 3: Semantic Channel

This level adds semantic support to lower levels. The support is provided to the Layer 1 in the level of a driver, and to the Layer 2 as an additional grammar for the EUD programming language. The support will be delivered by allowing XML syntax. In the driver the XML will allow to define meta descriptions of API calls, as well as meta description of a resource itself. In the EUD programming parser, the XML describing RPC calls will be accessible as function properties, but only treated as plain text, and no meta language parser will be implemented until the Layer 5. However, a programmer can choose to programmatically parse the meta

content. The layer is called “Semantic Channel” as it only implements a carrier for the meta language, without implementing tools for processing and interpreting, except for a a standard object called “tag”. The object “tag” is introduced into the layer 3 as a built-in type. Let’s consider a code below:

```
<example value=”demo”/>
var t: tag;
var s: string;
t.assign(“example”);
if t.exists then
 begin
  if t.has(“value”) then s:=t.get(“value”);
  if t.count>1 then s:=s+’,’+t.paramid(2);
 end;
```

The object tag has following methods:

- assign – specifies a tag’s ID to which the tag object reffers
- exists – returns TRUE if a tag was set in the script and FALSE if not
- has – returns TRUE if tag has a parameter of specified name
- get – returns value of specified parameter or empty string
- count – returns number of tag’s parameters
- paramid – returns name of a parameter by its index (0 is the tag id)

The first occurrence of a tag in the source code will register the tag and keep it until the end of the script. Any following occurrence of the same tag will replace the previous one, and the registered tag will be updated.

Additionally, layer 3 implements a built-in function called **RegisterGlobalTag**, which takes 1 argument of type string, to pass a specific tag id. All of the tags of that ID will be registered in the global memory space (managed by API Operator) and will not be removed after the script exits. Any subsequent occurrence of the same tag in any script at any point in time will not overwrite preceding occurrences,

but instead will be added into an array. The array can be browsed and any occurrence of the tag can be accessed by its index in the array. If the index is not specified, it is assumed that reference is made to the most recently recorded occurrence. There is another object provided to handle global tags, called globaltag:

```
var g: globaltag;
    t1,t2: tag;
g.assign("tag1");
t1 := g;
t2 := g[3];
```

In the code above, there are 3 variables: g of type GLOBALTAG, t1 and t2 of type TAG; The type GLOBALTYPE is actually an array of objects of type TAG, and therefore "`t2 := g[3];`" is possible. The code "`t1 := g;`" is equal to "`t1 := g[0];`". Additionally the GLOBALTAG implements 2 methods:

- length – returns the length of the array
- setlength – limits the length of the array

The SETLENGTH method will only take effect if its argument is smaller than the current length of the array, and can be used to cancel specified number of most recent occurrences of the tag. At the beginning of a script, the initial length of an array for a global tag can be saved, and at the end of the script all the recent occurrences of the tag that happened during the script execution can be canceled, by limiting the array length to the initial value. However, due to multi-threading there is a risk that using the described method can cancel some important occurrences of the global tag that happened within another script that was running in parallel to the current script. For a better management there are two additional methods of the GLOBALTAG object:

- Snapshot – to create a restore point
- Reverse – to cancel of the occurrences of a global tag that are coming only

from within the current script, and only back to the most recent snapshot

A snapshot is always taken automatically at the moment of calling .assign method of the GLOBALTAG object. While these two methods are advised to be used instead of SETLENGTH method for some cases, but sometimes it may be an intention of the user to limit the tag array regardless to the scope of the current script. For example a rule can be implemented using SETLENGTH method to reset global tag array periodically, or to limit a maximal allowed length of the array.

If a negative value is passed to the SETLENGTH method, the array will get limited from the other end, keeping only the most recent occurrences and discarding old ones.

3.5 Layer 4: "Company" model

A company has an organizational structure, in which an employee has a supervisor, who can have another supervisor and so on. There are departments, and sometimes inter-department positions may be found. Every department, every team, and every employee has a role. The role is always assigned from the top management downwards. But in the real life, an employee may change position within a company, may leave the company forever, or for a period of time and then return, can get sick, or take a leave, can be required to substitute another employee etc. Everything is being handled by the company internally in a vertical flow of decisions (from the top management). However for a customer of the company, most of that remains transparent. The customer has an interface to communicate with the company (an account manager, a clerk, a customer service, an email, etc.), and to communicate his needs. He needs not to worry how exactly the company will organize itself in a workflow to deliver him the results. He does not care which employees will do which parts of the task, or if some employee is replacing other,

who would normally be handling particular task. It is up to company's own problem how the work will be distributed and executed internally. For all that the customer cares for, is the result. Layer 4 implements this philosophy. Using Semantic Channel from the lower layer, a support for defining roles to resources is added. The System will analyze existing resources and group them according to functionality. The functionality will be examined by reading meta descriptions of RPC calls, and the system will try to find similarities between different RPC functions across different resources, then, it will try to create a mapping of how certain resources can substitute other resources in delivering specific type of results. This layer does not implement support for taking semantic descriptions of high level goals as input, but if a resource assigned to a particular RPC function is not available, it will try to automatically find a substitute and enable the execution of RPC function by alternative resource.

Implementation of Layer 4 is achieved by extending syntax accepted by the interpreter of the lower layer, so that it additionally allows for using XML tags within the source code. Consider a sample code for the layer 2:

```
var s: string;
s:=systemTimeStr();
TextOutXY@simple-lcd.com(10,10,s);
```

Let's assume that systemTimeStr is a function implemented locally on the API Operator, and returns plain-text encoded current system time. Let's also assume that the "simple-lcd.com" is a domain used by a simple, buttonless LCD screen that has an API Contributor component embedded and is hanging on the wall in the living room, so that the system can display content on it. Let's assume, that this LCD screen exports function called TextOutXY which displays a text at given position XY (Upper-left corner of the text area). The code above would load current time to

a variable s, and then print it on the LCD screen in the position X,Y. In the layer 3 we could add XML tags to add some meta descriptions, and the interpreter would compile it as valid syntax, then in the layer 4 we additionally have few standard XML tags and a set of standard functions for handling these tags:

```
<functional-tag value="current-date" functionref="systemTimeStr"
argcount="0" result="string" />
Var s: string;
s:=FunctionByFunctionalTag("current-date",0,"string");
TextOutXY@simple-lcd.com(10,10,s);
```

The first XML tag is called "functional-tag", and allows to group functions according to what are their purposes and number of required arguments. In the example above, the tag's value is "current-time", and ARGCOUNT is "0" which means, that the function does not require any arguments. The FUNCTIONREF argument contains the function's name. This XML tag will register the function "systemTimeStr" globally as delivering a functionality called "current-time" and not requiring any arguments on calling. And as returning a value of type tsring;

The second instruction is allocating a variable s of type string in the memory.

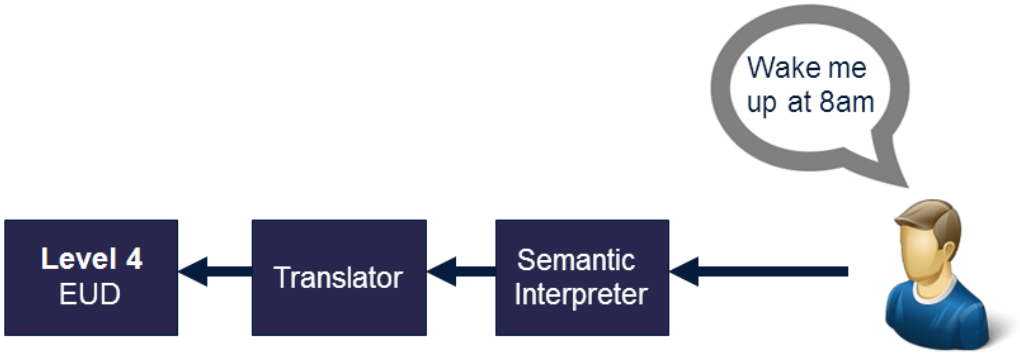
The third line of code, is a built-in function being a part of standard set in a system with Layer 3 implemented, and is used to call a function by specifying particular functionality rather than by calling a specific function by name. The call takes 3 arguments: the first is a functional-tag value, the second is number of arguments, that the function must require in order to be called, and the third is a type that the function must return. The call in the demonstrated code, will browse through all globally-registered functions matching the functional-tag, without any required arguments, and returning a value of type string. After determining a list of all matching functions, it will choose the first listed function implemented locally o the API Operator, or the first matching function that comes from an API Contributor on

the local host (relatively to API Operator) or just the first function matching from the list. If no matching function is found, the code will not take any effect on the value of the variable s. This mechanism allows for writing scripts without specific calls to specific functions, but with references to globally registered tags describing functionalities and calling formats for related functions. The system can take make a decision at run-time about which RPC to call in order to address requested functionality, and in special cases, if a specific RPC fails (network error or resource is unavailable), the system can attempt to deliver the result by matching an alternative RPC on an alternative resource.

3.6 Layer 5: High-level EUD

This level implements a semantic parser for XML input and a module for translating semantic descriptions of high-level goals into specific RPC calls of low level. In this layer the EUD programming is supported purely with XML. A code can also by a hybrid containing both XML and the lower layer programming language, in which case two parsers will be involved in executing it – the semantic parser from this level, and the EUD interpreter from Layer 2 extended but Layers 3 to 4.

In this layer an end user can call the system with requests like “Wake me up at 8am” and the system will try to translate this request into corresponding sequence of RPC calls to relevant resources. In the case when the high-level goal is not possible to be interpreted, the system will interact with the end user to add missing semantic definitions needed for parsing the request. Let’s consider a diagram below:



The Layer 5 assumes integration with an external semantic interpreter (is only a conceptual reference design), which can parse the human language message and break it down to semantic description of a goal, referring to a local database storing semantic definitions that are collected over time during interaction with the end user, or which can be fetched from other databases. If there is any definition missing to solve the message, and if it cannot be fetched from a remote database, the system will create an alert, to which the end user can respond and provide with additional definition. Only upon gathering all the definitions required to solving the message, the syntax parser can send its output to a translator that will convert it to a source code accepted by the layer 4 interpreter for the EUD programming.

This layer is the leas complete in specification as it is a conceptual reference model for adding semantic layer on top of the proposed system, where the semantic interpreter is not a part of a design, but is expected to be provided by 3rd parties.

3.7 Current implementation

Currently the implementation has been accomplished to cover Layer 1 and partially Layer 2. The implementation has been made in Delphi dialect of Object Pascal language, and compiled for Windows family operating systems. The source code with precompiled demo is attached to this thesis on a CD. The original feature of this implementation is that the virtual hosting for variables and function calls functionality is combined with the

parsing functionality.

3.7.1 libEngine.pas

This library is introducing most important classes for embedding API Operator component within existing code. It is also initializing key components to set up the API Operator environment automatically at the runtime, in the moment when the library is loaded. The code is very short:

```
unit libEngine;
interface
uses SysUtils, libSupportedTypes, libHostedVariables, libHostedFunctions;

Var
  SupportedTypes              : TSupportedTypes;
  GlobalVariables             : TVarTable;
  HostedFunctions             : THostedFunctions;


implementation

Initialization
 //Supported Types:
   SupportedTypes := TSupportedTypes.Create;
   SupportedTypes.Add(VTString.Create('string'),stfString);
   SupportedTypes.Add(VTInteger.Create('integer'),stfInteger);
 //Global Variables:
   GlobalVariables := TVarTable.Create(SupportedTypes);
 //Hosted Functions:
   HostedFunctions :=
THostedFunctions.Create(SupportedTypes,GlobalVariables);
Finalization
 //Hosted Functions:
   HostedFunctions.Free;
 //Global Variables:
   GlobalVariables.Free;
```

```
 //Supported Types:
   SupportedTypes.Free;
end.
```

The important code is inside the **INITIALIZATION** section. **SupportetTypes** object defines data types supported by the framework and their lexical identifiers. In the current implementation there are only two supported types: **string** and **integer**. End-user might implement conversion routines for encoding additional types within existing ones, for example, encoding **float** values as **string**, or **Boolean** values as either **string** or **integer**. Naturally, more types can be added to the native support, and they would need to be initialized in this library as well.

The **GlobalVariables** object is managing the virtual memory and variable allocation within the API Operator. The constructor of the class requires reference to previously configured **SupportedTypes** object, so that the memory manager knows what data types can be allocated as variables.

The **HostedFunctions** object is for allocationg and managing dynamically exported RPC calls. The class' constructor requires reference to both **GlobalVariables** and **SupportedTypes** objects so that it can carry out type control and variable substitution during the interpretation of the RPC calls, which are plain-text encoded instructions. It is important to note, that these objects will only host functions and variables in the local memory space, so only functionality exported from within the application hosting the API Operator component. A different set of classes is responsible for managing actual remote calls.

3.7.2 libSupportedTypes.pas

This library provides with classes needed to implement natively supported data

types. Every data type to be natively supported is a class inheriting from **TSupportedType** mother class. One important element for any natively supported type is built-in conversion to and from the plain-text string type. This are virtual methods, that have custom implementations for each supported data type. These methods are used during the interpretation of the RPC code, when the plain-text lexems are passed to check whether or not the conversion is possible, which is the actual delivery of type control functionality.

3.7.3 libHostedVariables.pas

This library provides with a class **TVarTable**, which allows for allocating memory for storing data of natively supported types, as variables indexed by plain-text identifiers. Each virtual variable is encapsulated within a very simple record:

```
TVarInfo
 = record
    TypeFlag     : TTYpeFlag;
    TypeID       : widestring;
    Value        : TSupportedType;
  end;
```

that holds 3 member fields: **TypeFlag**, which is code-level flag assigned to a natively-supported type, **TypeID**, which is a plain-tex unique identifier of the data type for use on the RPC code level, and the **Value** field, which is a reference to an object of **TSupportedType** class that implements interface to access (reading, writing and conversions) the actual data, and which itself allocates the actual memory for storing the data. The **TVarTable** class allows for accessing variables and their data by the **TypeID** and is being a part of the interpreter itself. In other words, not only it serves as an interface for accessing the hosted variables, but also provides with basic set of methods for parsing

plain-text encoded lexems for verifying whether or not they refer to actual variables.

3.7.4 libHostedFunctions.pas

This library provides with classes for hosting RPC calls, even though in the case of API Operator component they also host locally exported functions, that do not require any network traffic to be executed.

Important low-level class in this library is **TFunctionPrototype**, which implements support for custom formats of function calls, including function identifiers and description of required arguments (if any). This class allows for creating a prototype for a function call at the run-time. It is contained by a higher-level **THostedFunction** class that not only allows for accessing the interface for executing the actual call, but also implements methods for parsing the plain-text encoded function calls. Above this class, there is a higher-level **THostedFunctions** class which manages a collection of **THostedFunction** objects. A very important definition from this library is **TExternalCallEvent**:

```
TExternalCallEvent = Procedure(const FunctionName: widestring; const Arguments: TVarTable; const ArgCount: cardinal; var FunctionResult: widestring; const ResultType: widestring; const RessultFlag: TTypeFlag)of object;
```

This is a procedural type, which define a generic and universal wrapper for exporting ANY function or procedure (function without result, void in C language) from the host application. This is not however, the final data type that the developer uses for coding wrappers, as there is a higher-level simplified version defined, that will be discussed later. The definition in this library requires few arguments in the wrapper:

FunctionName – is a plain-text encoded unique identifier for a function

Arguments – is a reference to the local object of TVarTable class allocating

memory for storing and handling function arguments, that exists only throughout the call of the function, and is being released after the call executes.

FunctionResult – is a string variable for returning plain-text encoded result value (it will be later converted to a proper type that is natively supported)

ResultType – is a plain-text encoded identifier of the supported type of which the function result is expected to be

ResultFlag – is the code-level flag with the same purpose as the ResultType, but easier to use in the statements and expressions, as it is a simple type comparing to string.

3.7.5 libErrors.pas

This unit contains error codes that the built-in interpreter can return to end user via the RPC gate. Additionally it contains few functions shared across other modules for parsing of plain text and HTML, as well as some system-defined constants that are the standard setup of some core configuration of the system.

3.7.6 Server/client vs. Operator/Contributor vs. Flavors

This might be a confusing portion of the architecture, if not explained. Essentially neither the API Operator nor API Contributor component is determined to be strictly corresponding to either a server or client role. Either of the two components might incorporate multiple servers as well as multiple clients within. The only certain thing following the system specification is that the API Operator will at least have one active server opened, thru which API Contributors can announce their presence in the network by default, and thru which an End-user default interface (RPC gate) is accessible via http protocol.

The term API Operator and API Contributor only refers to the role in the

system, according to the earlier description in this chapter.

The API Contributor may come in many different variants, so called flavors. The simplest flavor is only incorporating a Client connection, and such API Contributor will use pooling to check with the API Operator if there are any pending RPC requests. When API Operator executes RPC via this flavor, the timeout delay is used for the API Contributor to pool for awaiting RPC calls, and for the second following pooling that will pass the result statuses of the previous queue of RPC calls. This flavor is dedicated to embed API Contributor components on devices on which CPU usage saving is crucial. Pooling saves lot of CPU usage and therefore also power usage when comparing to a server connection that is left open..

Other flavor is incorporating a server connection, and RPC calls invoked on this flavor of API Contributor have fewer critical stages during which a connection/network error may interrupt the normal RPC flow (RPC call + ACK roundtrip). The API Operator will only need to open a client connection to the API Contributor once, and together with submitting the RPC call fetch the result status, just like a web browser is requesting a webpage directly after submitting GET or POST request with specified headers and query string parameters. The ACK can be obtained within a single roundtrip, versus a double roundtrip in the case of the first introduced flavor. Also the tolerance for delay may be lower, as there is no need for queuing RPC calls and waiting for the API Contributor to fetch them via pooling.

Additionally to the above obvious flavors, hybrid flavors are also possible if a specific application needs it.

Although the second described flavor is less vulnerable to connection/network

errors during the RPC call roundtrip (RPC call + ACK), but it also exploits the API Operator to a greater degree, requiring more client connections to be dynamically managed, therefore each flavor has its own advantages as well as disadvantages.

API Operator also comes in multiple flavors, however in this case it more often refers to the amount of server connection opened for supporting different protocols, like HTML, XML, JSON etc.

3.7.7 libServerMod_HTTP.pas as a flavor of API Operator

This library defines classes for the default flavor of API Operator component, which uses HTTP protocol. This flavor incorporates **THTTPServer** class responsible for adding active server connection that the end-user will be able to directly access via web browser, on a custom port. This flavor implements methods of the http server object with a basic authentication mechanism, that will require end user to use a login and password to access the default interface (RPC gate).

Additionally, the very same flavor supports a plain-text based communication, after not finding valid HTTP protocol header in the incoming connection. This is used by default by API Contributors when connecting to the API Operator, and allows for providing both the user interface and the default RPC channel over the same active service.

Use of this flavor implies use of a CSS file in the home directory of the API Operator host application, that will be used (if present) to load styling for the HTML based user interface. The CSS file has to be called NETPASKAL.CSS, and the naming comes from an older programming project that the discussed system was implemented upon, that dates back to the project for my Bachelor

thesis at Adam Mickiewicz University in Poland.

3.7.8 libMemberTCPLite.pas as a flavor of API Contributor

This library defines classes for the default flavor of API Contributor component. It incorporates a client connection and a timer, that will trigger pooling to the API Operator component over TCP/IP (either over network or local host). It is the lightest flavor for the API Contributor, and uses the plain-text communication over the default channel, through which the HTTP access to the default end user interface is also provided. Any other flavor of the API Contributor would inherit form this flavor and use the default channel to set up a custom one(s) and hand over the further communication.

3.7.9 Extending flavors

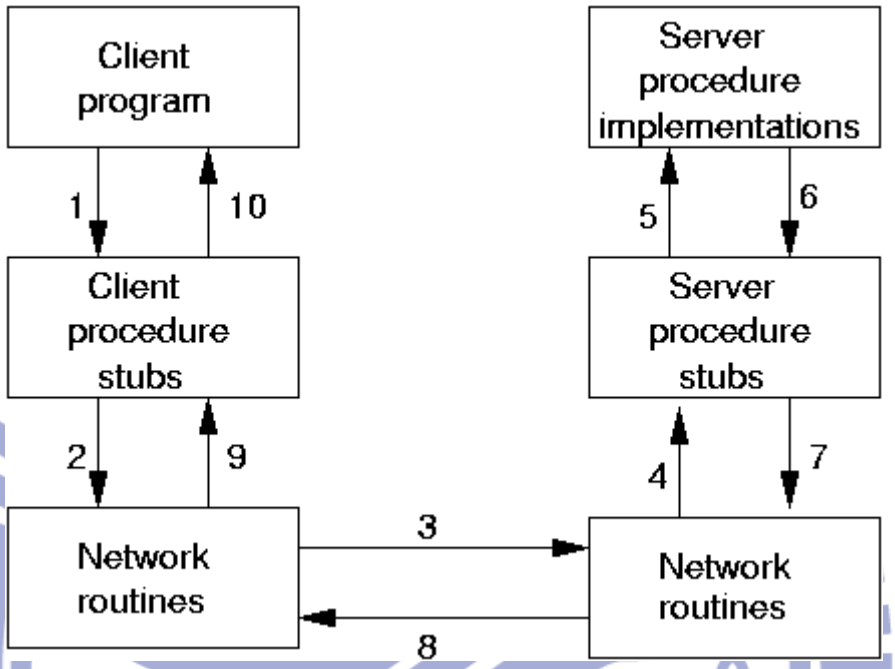
Flavors might be added as modules (for example as DLL libraries for Windows-based API Operators) and can add communication channels and protocols over which RPC transactions can be carried out. For example, a flavor enabling IPC messaging can be added, for handling RPC transactions via IPC channel rather than network connection via sockets. This may be a faster alternative for handling API Contributors residing in the local host in relation to API Operator. Above the level of the flavor's internal implementation there is no difference and the RPC transactions become scalable transparently.

3.7.10 uAnLex.pas

This library implements a Lexical analyzer used in the process of parsing RPC calls and EUD programming scripts. The code was originally implemented for my Bachelor's thesis at Adam Mickiewicz University in Poland, and somewhat upgraded since then for more convenient use and less memory consumption.

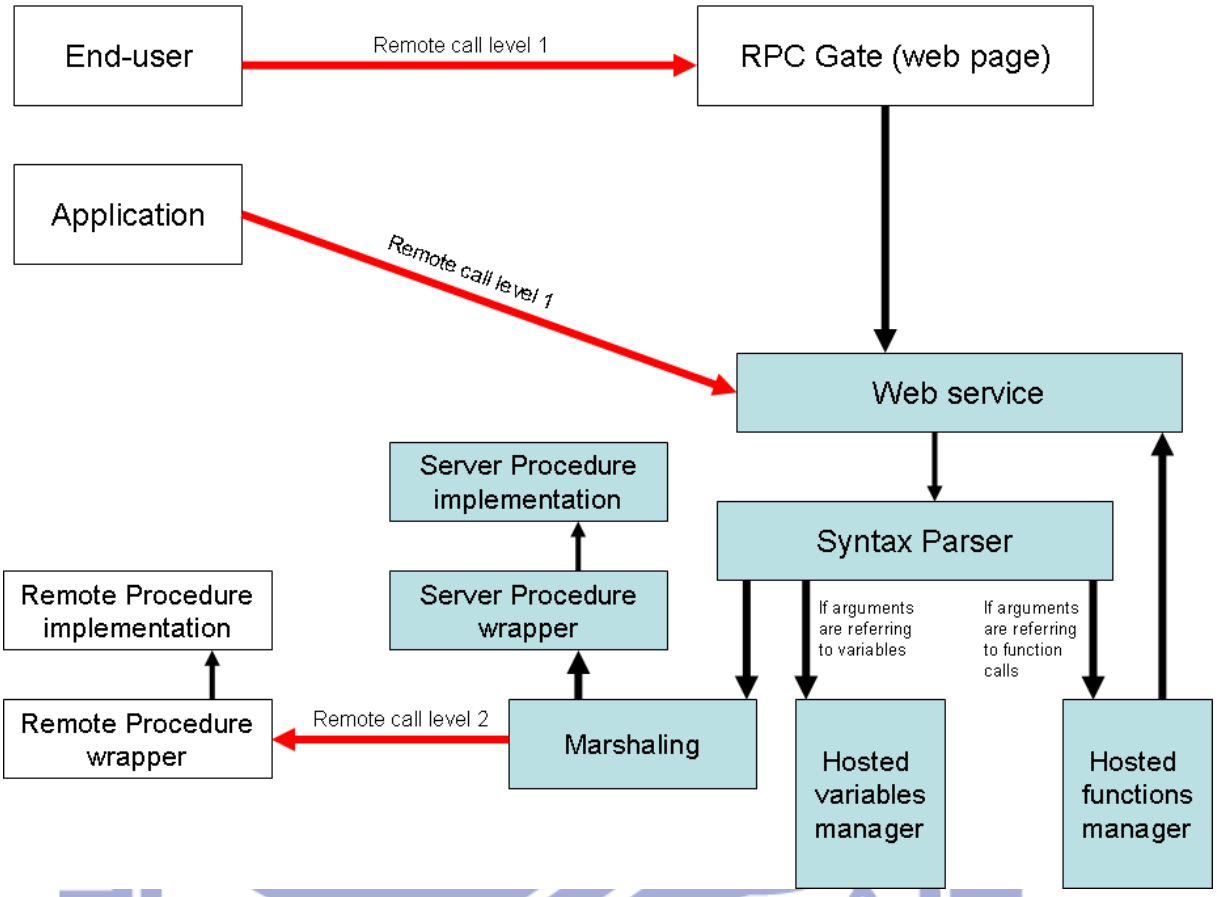
3.8 Non-standard approach to RPC

The proposed design changes the standard RPC model quite a lot. As a reference, below is the simple diagram for the standard RPC flow:



It shows clearly that the RPC calls are handled by the server and on the server, and the remote call happened one time in two directions, as a roundtrip from Client to server (call + ACK).

Proposed architecture is greatly extended version of the standard RPC, but the extension remains transparent to the caller. The main improvement is that the RPC call is only being managed (also validated) by the server but can be handled also outside of the server by other machines that are remotely connected to the server. The simplified flow for the RPC in the proposed design would look like this:



The red colored connections are the remote transactions (happen over network). In contrary to the standard RPC model the server might not be the terminal point for RPC execution. An RPC might require a subsequent remote transaction over the network to a remote RPC implementation – remote in relation to the server. The teal colored components are all implemented within the API Operator, after the web service receives an RPC code (it might consist of a program block with multiple RPC calls) it is parsed by a built-in interpreter. When a function call is found, the syntax parser evaluates given arguments. If there are any references to a variable the parser calls Hosted Variable manager to fetch the actual values and substitute the argument with it. If there are any references to a function, the Hosted function manager is called, to check if a function is listed and if it returns correct type, then, a web service is called once again from within an iteration to solve the function call, until the result is obtained and can be passed back to the Syntax Parsed for substitution. Finally, where all the references are solved, the

RPC call is being marshaled and sent over network to a API Contributor component, that will trigger a remote function wrapper, that finally triggers the actual implementation of the function The API Contributor component can reside locally on the server, or remotely in the network, in either case, yet another RPC call over the network is being issued, which is an additional level not present in the standard RPC model. The API Operator itself can also share functions that are implemented within itself, in such case, the function call can be resolved directly through hosted variables manager and hosted function manager. Each of the connections on the diagram is actually bidirectional and if an error occurs at any point, the flow is broken and returns the error code back to the web service over the shortest reversed path.

This approach for addressing RPC is actually a multi-level alternative, extended version of the standard RPC model.

4 Solution evaluation

4.1 Addressing defined targets

The first target introduced in Chapter 2 were end users. This solution will potentially improve their lives in the way that they will gain more freedom in how they want to use all the software and hardware resources they happen to own. The proposed system gives them more control over devices, and a way to go with their creativity to implement various DIY customizations to their technology-enabled daily life. Furthermore, satisfying their user experience with the solution-enabled devices and apps is also a goal, which can be reached if hardware vendors, software developers and system integrators see the business opportunity in building their own products and services on top of the proposed system.

The second target group is made of businesses. Where the improvement coming from the use of proposed system is considered for the existing product and services, it is much faster (and cheaper) to add RPC functionality to existing hardware and software products by using a 3rd party middleware and linking pre-made libraries than by designing and implementing communication layers, protocols and parsers in house. Although alternative middleware and libraries might be considered instead, but the proposed architecture is especially designed and dedicated to be integrated into the existing code and hardware with the minimal time and effort needed. Another competitive edge for businesses comes from lowering the costs of implementing future upgrades of already compliant products and services. The proposed architecture is especially designed to allow for adding more functionality to compatible products without the need of inside the box modifications. A product with a constant number of features that are exported to the be used over proposed system, can be upgraded to virtually infinite number of new features even without any modifications needed in the original hardware nor firmware, by just providing a new software independent from the product itself. Furthermore, the existing product can be integrated with features of other compliant products without the need of modification of neither one. This also makes a great added value to a product, as from the end user's perspective, because even after there is a new successor to the product, or if the product support gets discontinued it is still possible to create updates for it or to dedicate it as a resource for the network. In this architecture there's virtually no end of life for a product. Even if the original provider (a vendor, manufacturer, etc.) goes away from the market, or stops providing support, any 3rd party can still provide updates, and the product can continue to be of use on and on. Lot of end users keep some of their old devices even after upgrading to newer ones. These old devices, even after not being used no more, are still fully

functional. Finally they too get thrown away, given away or recycled. But if only they could be dedicated as resources to newer generations of products, they would surely continue to serve their owners or would otherwise be valuable goods on the second-hand market.

If businesses create revenue on products upgrades (like Apple does), it might seem at the first glance as a threat to new sales if old product don't go into the end of life stage. Fortunately in the real life the advancement of technology will still push customers to purchase new products, while the old ones, instead of being thrown away, can continue to serve their users but in fulfilling different, lower level demands.

Third target group is the industry. Special consideration has already been given to the manufacturing industry (see Chapter 2). There are two primary ways how the proposed system can bring benefits to hardware manufacturers. Both are related to the OEM market. The first benefit is related to their current client base. Currently, where there are many OEM suppliers available, the brand owner will expect products rich in features ready to put their logo on and to market it right off. This is already how things are happening with lot of consumer electronics (CE) like Smartphones or tables as an example. Nowadays it is possible to order a fully functional tablet pc from a supplier, with requested logo printed on the product, paper documentation and boxing, that can be sold to end-users right off the shipment container. Still, lot of brands will create their own customizations. In tablets for example a vendor will by default make a custom ROM (and not only the brand company, but even a reseller can make customizations, like Telecom providers are often doing). Selling a product that is compatible with the proposed system would allow these brands to even more customizations. In fact, an OEM could start selling white boxes with sensors and peripherals only, while their customers would implement actual features for the end users outside of the box. The

second benefit is that OEMs can start selling directly to the end-user market (or through distributors) without any additional spending on things like branding, marketing etc. Every single component from a current hardware spec can be sold separately to end-users as dedicated for DIY. A typical piece of electronics, if it is just a component, not a standalone product, has little value to an end-user, unless he is an electrical engineer or a hobbyist with some EE skills. However, if the single component is compatible with the proposed system it immediately becomes useful for any end-user regardless of his technical skills or their lack. A user who is not technically inclined, can bring the component home and create a self-made solution out of it in minutes. Alternatively, the end user can use a 3rd party software able to use newly added components as kind of Plug-n-Play resources. If a high level goal given by the user to the system cannot be realized because of some missing resources, the user can simply go to a nearby electronic store, grab few cheap, no-brand components that will just do, bring them home, and let the system to detect these added resources, configure them, and start using, to rich the given goal.

Finally, the 4h target group are the developers. Apart of everything that has already been covered and that has already shown possible benefits to the developers in terms of exploring new market share, there is a set of benefits specific to the process of developing software. Targeting all the previous groups greatly depends on high involvement of developers working over the proposed architecture. What are benefits for them to choose working on this specific solution rather then implementing some high-level integration and communication solutions without it? Some most crucial benefits include:

- very easy integration of the system with existing code (linking a library and very few lines of code to export functions) in the Level 1 development (Ch. 3.2)

- Even without consideration to a wider problem of general inter-device and inter-application integration and communication, the proposed solution can still work great as easy way to implement RPC, network communication or CGI/Web services features to existing code for specific and limited use
- On the Level 2 development (Ch. 3.2) a native support for scripting programming language syntax – after all the communication, protocol and parsing has been provided, the further development on top if it does not increase complexity of code. The added layers are transparent, and the RPC handling works seamlessly, so the code looks like if no remote communication was happening
- Adding further levels on top of the level 2 maintains the complexity of the source code, as each additional level transparently hides all the levels beneath. In fact an integration between different levels is possible

4.2 Meeting benchmarking criteria

Below is the evaluation of the system measured through a benchmarking model (Chapter 2.4.6). For each criteria in the benchmark a score is assigned that can take one of 3 possible values: -1 if the system is graded as weaker in fulfilling the criteria than chosen competitor, 1 if the system is graded better than the competitor, and 0 if the winner is unclear:

User Experience Unit for benchmarking		
Intuitive UI	Compared against: CORBA, AJAX, Web Service; The advantage of the system is that a simple GUI for the end-user is natively built-in. Compared solutions require a developer of a web application or web service to build the GUI himself.	1
Learning Curve	The set up of the system requires an end user to download an API Operator software and install it on a PC or mobile device (alternatively by purchasing a mobile device, with built-in API Operator). After the setup, end user can just power on compatible devices and the system will be able to detect them and configure. Each detected device will have a standard, native GUI created for listing and remotely accessing its exported functionality. Optionally	1

	here can be a 3rd party GUI provided as well. This means, that regardless of how many different UI schemes might exist among different product makers, there is one native UI that the end-user can always choose if the one shipped with the product does not seem intuitive enough.	
Scalability	The process of installing new devices under the system is simplified. The only information that end-user needs to look up in the product's manual is how to hard-reset the device. The rest of the configuration is being handled automatically by the system, but interactively with the end user. A new device might have built-in facility to point it to the an API Operator from the End-users network, and if there are many API Operators in the same network it does not matter to which one the new device connects first – once connected the auto configuration will happen. Even if the device does not provide a physical facility to point to an API Operator, there is a natively supported procedure of installation (see Chapter 3). A device can be configured (authenticated) to multiple networks and switch between them seamlessly (or work under few networks simultaneously).	1
Applications base	The system architecture does not implement application store, but has a mechanism similar to GET-APT from Linux to fetch User Level applications from the Internet. Also, the system distinguishes between official and 3rd party (or trusted and non-trusted) domains to fetch applications from. Existing home automation systems do not yet offer this type of channel for distribution of user applications.	1
Affordability	By the will of the creator, the API Operator software is licensed as free of use for non-commercial use. No dedicated hardware is required (might be available as optional).	1
SDK Unit for benchmarking		
Package	SDK consists of non-redistributable edition of API Operator (free to use for testing), and set of binary shared libraries for supported OSes and programming libraries with source code for supported programming languages and platforms. There is no default editor in the current spec.	-1
Learning Curve	The SDK consists of a tutorial and sample programs, and it is anticipated that a developer can learn to write first programs for the systems within a day. The only thing to learn is to how to attach a library to existing code, and how to write a wrapper function. Past that, everything else is just programming in developers own	1

	language and programming convention. There is actually no new programming language to learn, just a basic knowledge of Pascal-based simplified convention of function call format for exporting functions.	
Portability	There is no actual need for porting, just enabling the compatibility. After a function wrappers are added and exported, it will be transparent for functions within the existing code whether they are being called locally or remotely.	1
EUD support	For encouraging EUD, a simplified scripting programming language interpreter is natively built-in. Also there is a native web interface allowing for execution of the scripts. End users can write simple scripts without need for SDK.	1

4.3 Proof of concept

The ultimate way to challenge a design is to turn it into an actual implementation. This chapter provides a description of an early prototype, a demo that puts it into the test, and a case study of using implemented functionality within an project for commercial use.

4.3.1 Early prototype

The first prototype implements a functional API Operator and a sample API Contributor. The prototype only implements the RPC Channel, and does not implement the Meta Channel. The API Operator implements a basic HTML interface for publishing the distributed API and for calling it. EUD support is limited to calling functions and declaring variables, but without support for loops, conditional statements and few other features.

4.3.2 Life demo

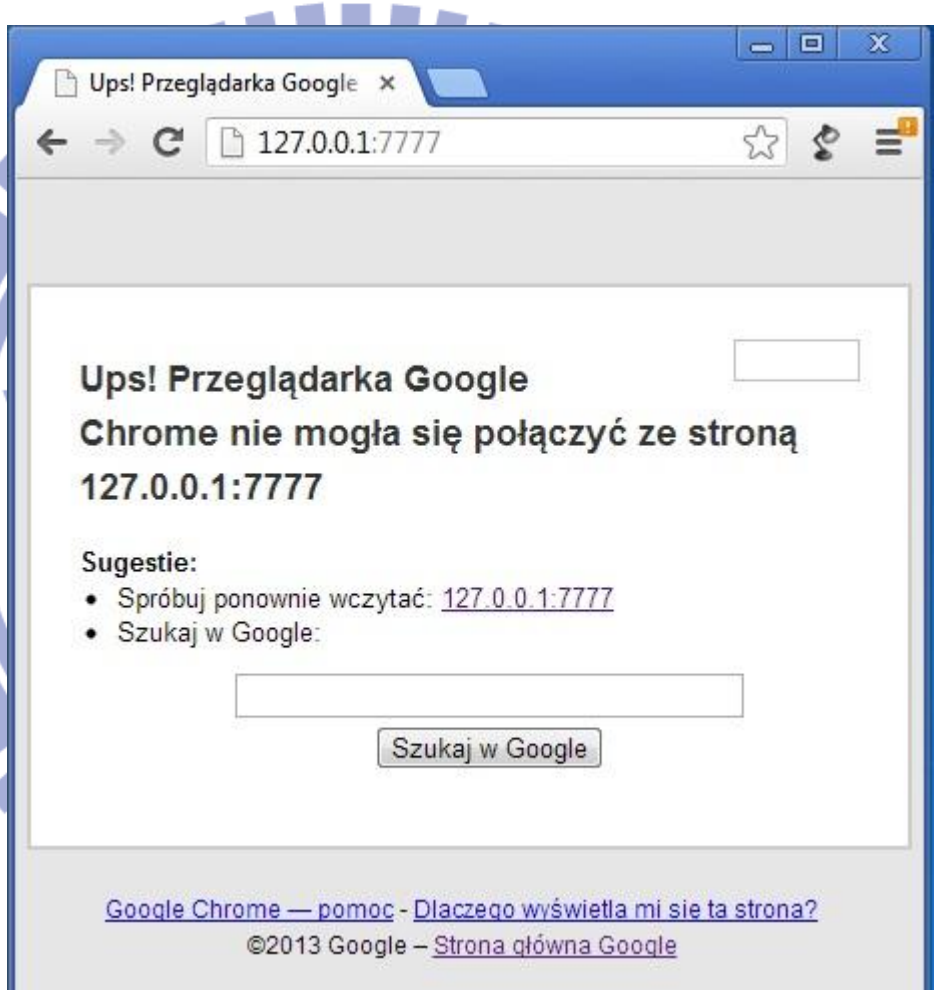
The life demo is dedicated to showcase a working prototype in action during the official defense of this thesis by the Master Thesis Examination Committee. The demo will demonstrate the implementation of Layer 1

and 2. Important features covered by the demo are:

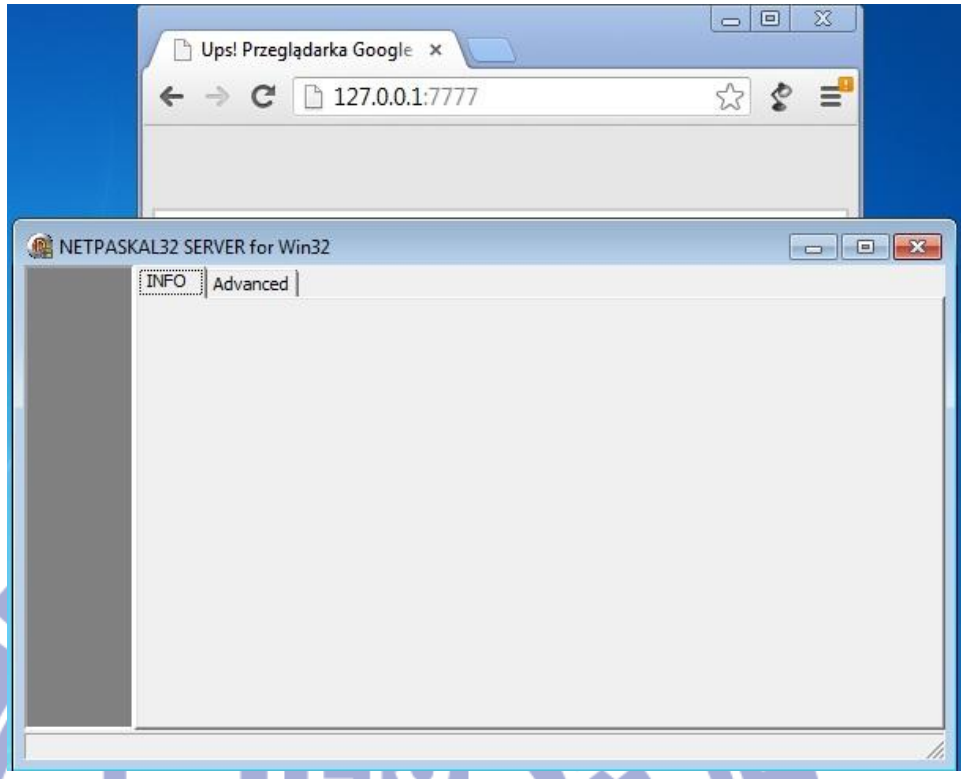
- User Interface (webpage based)
- RPC and virtual memory

The demo is prepared to follow steps described below:

Connect to 127.0.0.1:777, where the IP address is the local host and 7777 is the default port for the API Operator to provide GUI over (API gate):



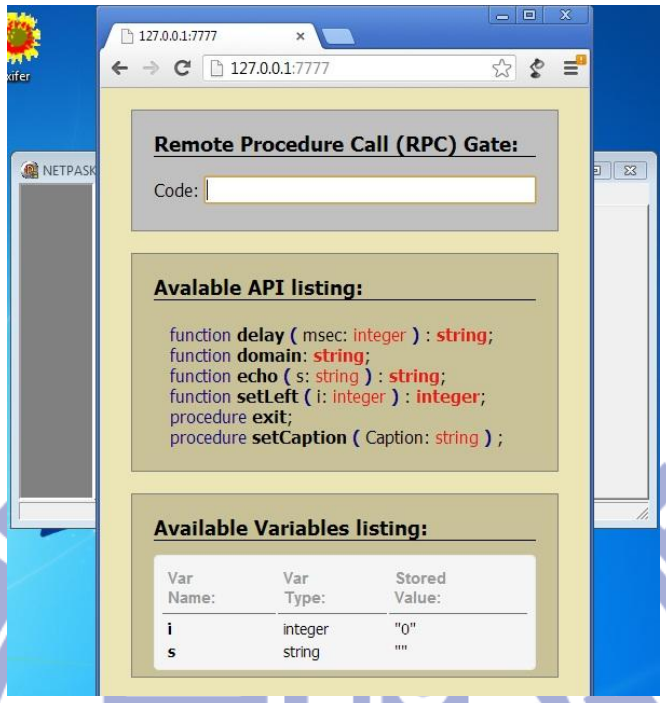
The browser will return PAGE NOT FOUND error, as the API Operator is not active. Let's launch the API Operator provided with NETPASKAL32.EXE from the attached CD:



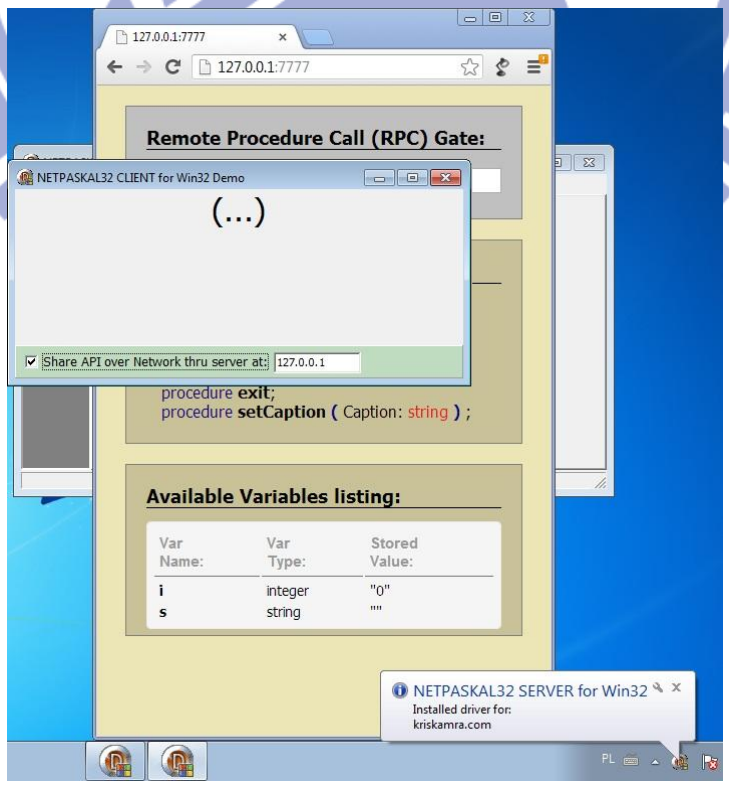
Above is the API Operator application running, let's refresh the browser:

The API gate authentication page shows up, for the demo purpose both the login and the password are "test". After logging in the default interface will show up, that is also presented in the Appendix 1, in the

demo the interface looks as follows:

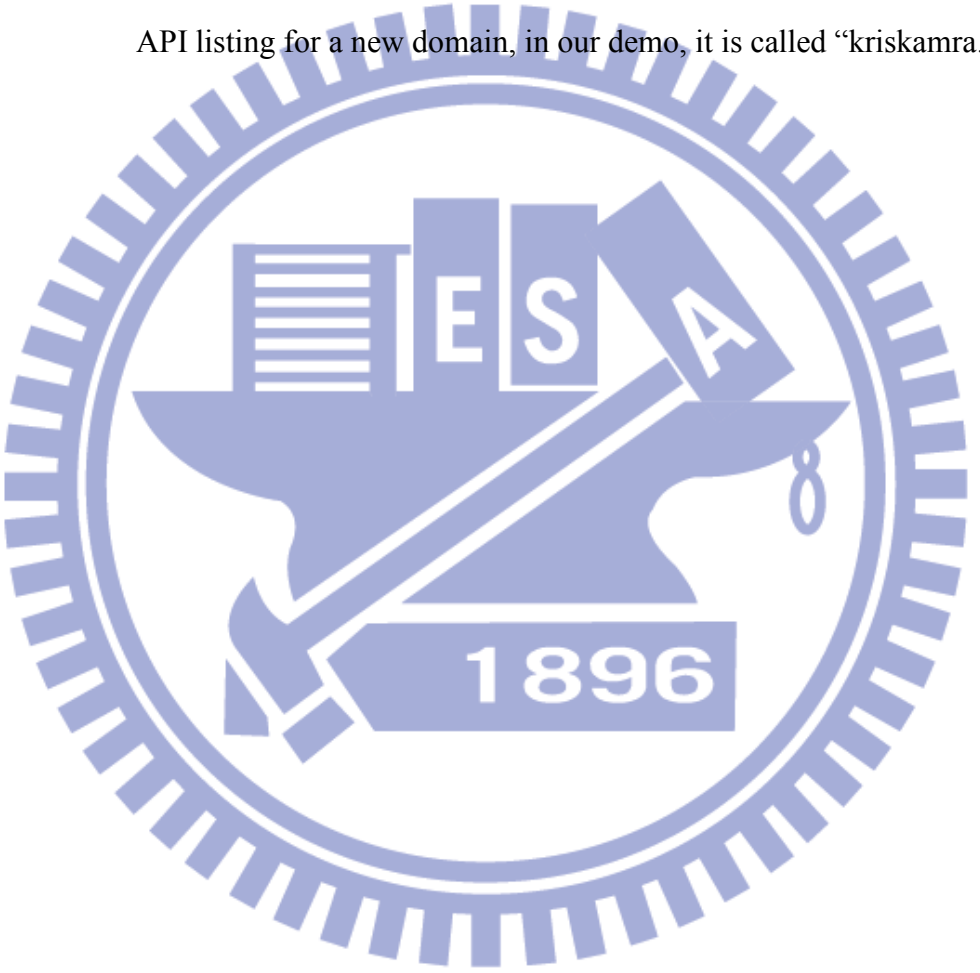


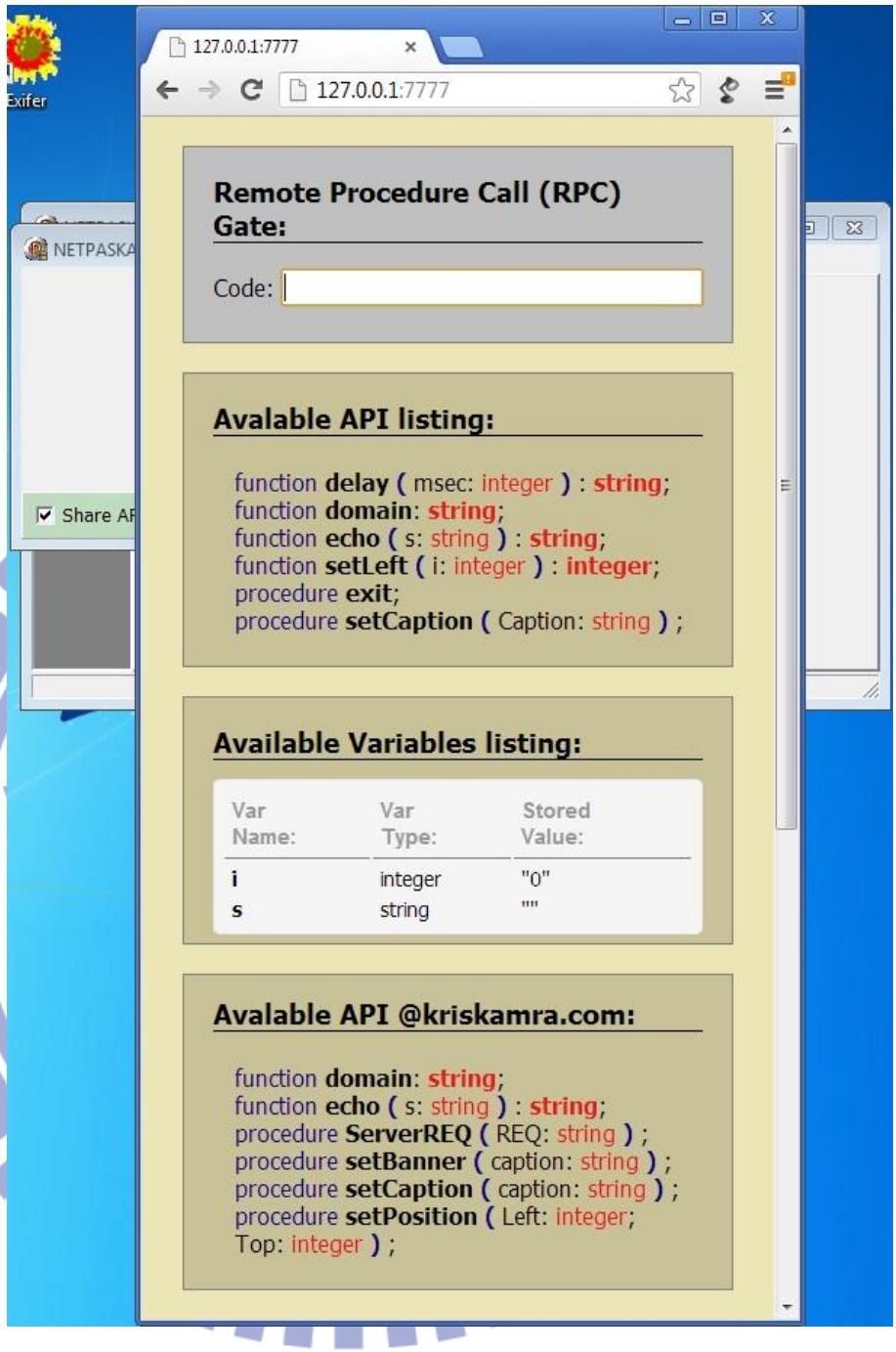
Now, let's launch an API Contributor via RPCCLIENT.EXE from the CLIENT folder on the attached CD, and apart for seeing the window of newly launched application:



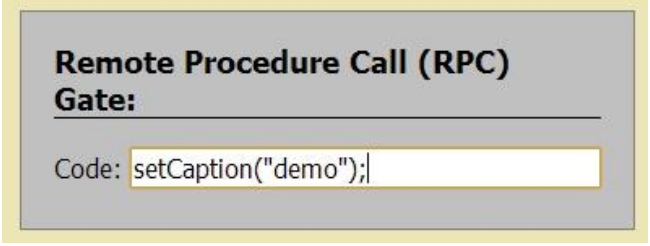
the important to see from the screen is the bubble-window notification in the bottom-right corner, that shows the detection of a new resource and automatic process of driver installation (refer to Chapter 3), which in the case of this demo, is delivered to the API Operator by the API Contributor itself:

After refreshing the browser, we will find out, that there is a newly added API listing for a new domain, in our demo, it is called “kriskamra.com”:



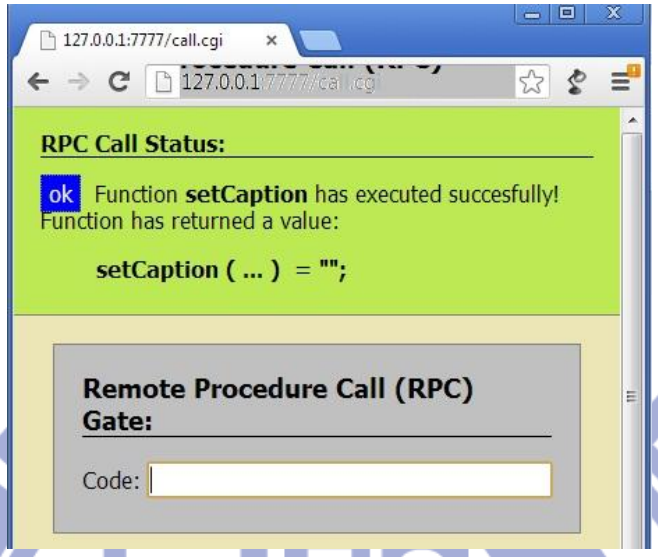


Let's make our first use of the API gate:

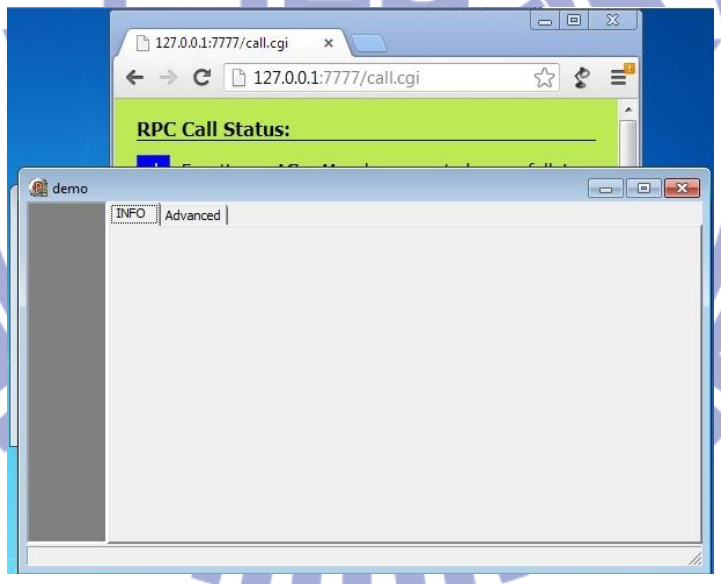


The function setCaption is listed both in the API Operator itself and under

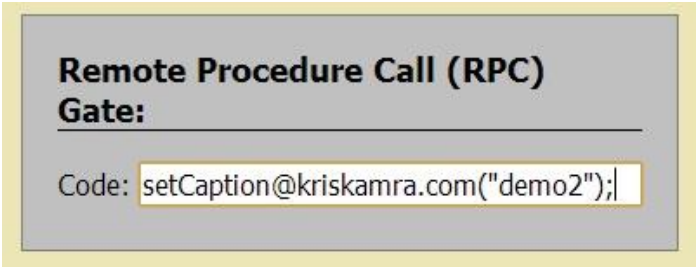
kriskamra.com domain, after submitting, the API gate will show the status above the RP gate:



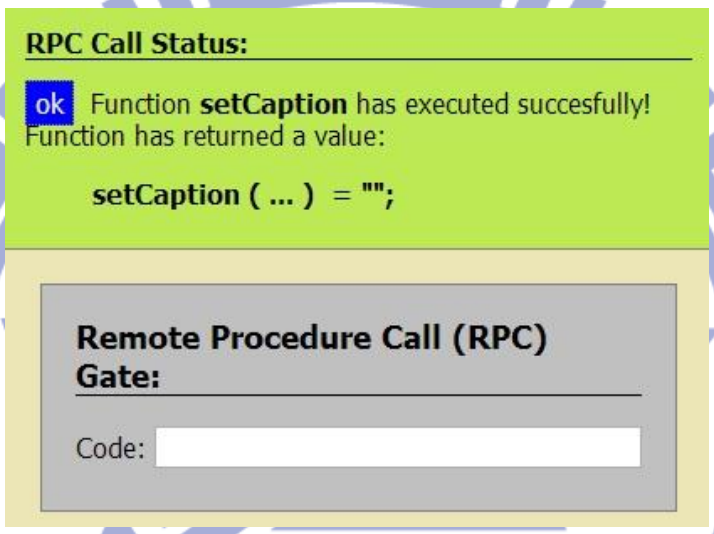
And the caption of the API Operator application will change:



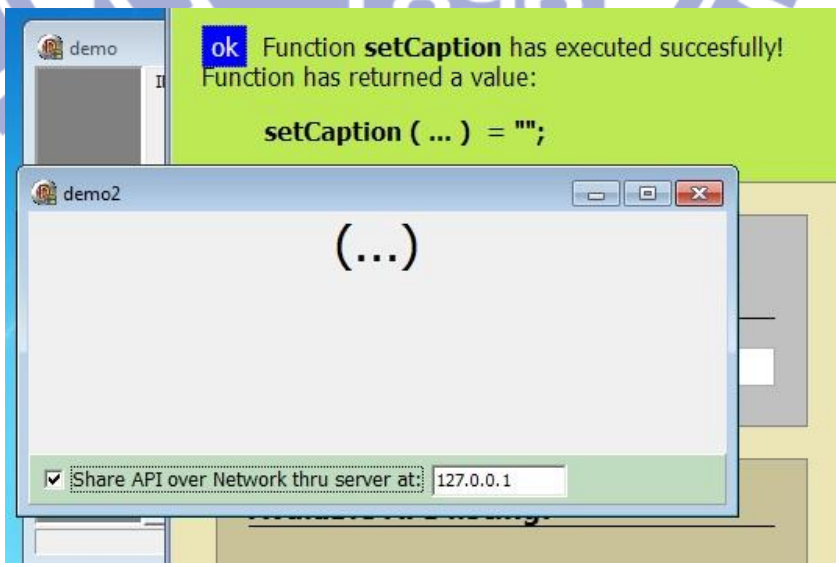
Now let's see how to call the setCaption function under from the API Contributor's listing:



The call contains the domain name after the function name, followed by the "@" symbol, which logically implies that we want to call setCaption function "at" the kriskamra.com domain:



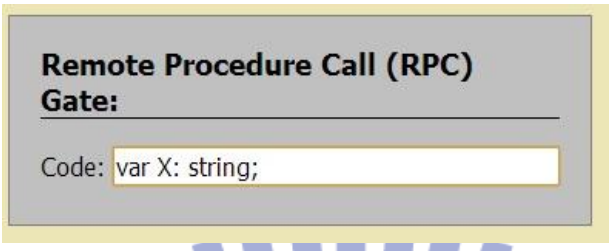
The RPC gate will accept the call, and as a result we will see:



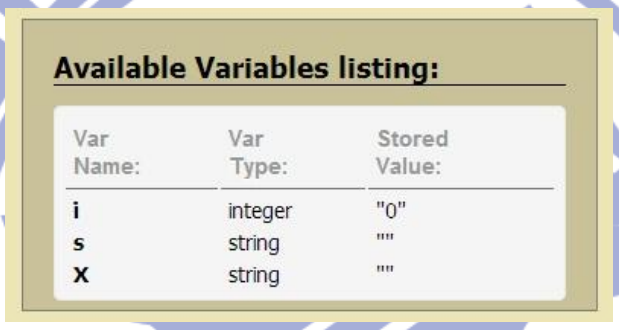
The Caption on the API Contributor got updated via remote invocation of its API function (remote in the sense, that the TCP/IP protocol carried the

RPC call over the network, even if the destination address was on the local host).

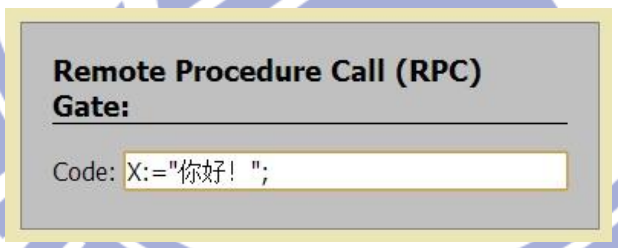
Now let's declare a new variable in the shared memory:



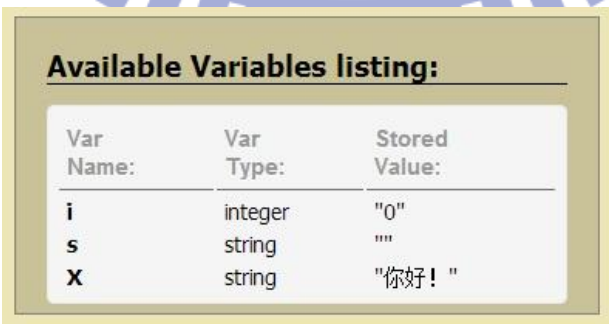
This will tell the API Operator to allocate a new variable of type string and call it X, here is the updated Variable listing from the API gate:



We can now assign any value to the newly created variable:



And new value will be applied and reflected on the listing:

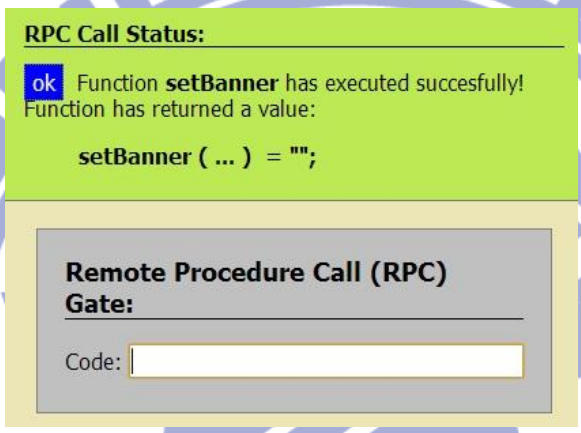


The system allows to assigned values to variables not only via constans but also through references to other variables and function calls. In the

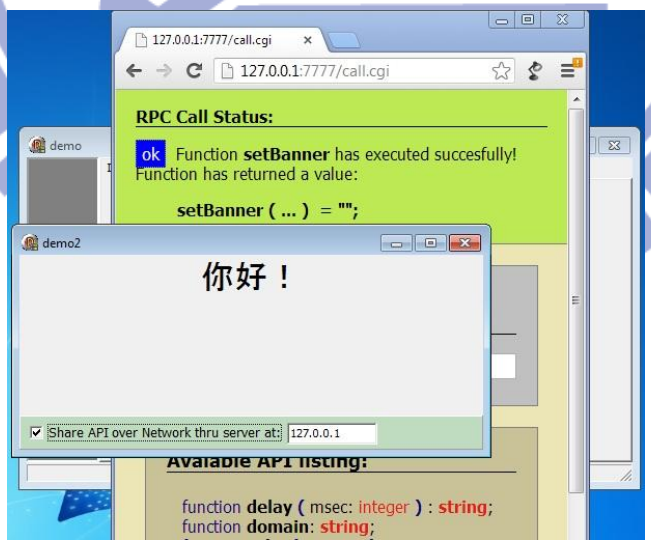
implemented demo only constants and references to variables will work.

Now we can invoke remote procedure setBanner passing variable X as the argument:

The RPC gate will accept the call and return status message:



Now we can see the effect on the API Contributor's side:



This is the end of this simple demo (Sources and binaries provided on the attached CD).

4.3.3 Case study

Soon after successfully implementing the first working prototype, also the first use of the implementation within a commercial project was possible. The project was a restaurant system with features like

- employee database and check-in/check-out interface (working hours registration)
- restaurant's menu editor
- seats status manager (displays seats availability)
- Digital Signage player

The prototype implementation was used in that project to add some extra functionality:

- remote food ordering (via web browser) for the customers
- remote control of the music playback for the employees (so they can control music in the restaurant from a mobile device)

The project was made for 誠食館 (Chengshi Guan) restaurant owned by Han-Chih Hsu with 2 stores in Hsinchu: 誠食館文化麵食，新竹市民生路 109 號 (No.109 Minsheng Rd., Hsinchu City) and 誠食館創意麵食，新竹市學府路 3 號 (No.3 Xuefu Rd., Hsinchu City).

Using the prototype solution allowed to add new features, without the need of implementing network connectivity, protocols, http-sessions, and such. A web script renders a dynamic html page with restaurant's menu. User can check and un-check items and adjusts quantities. When user is finally ready to submit the order, the Javascript simply generates formatted text string containing the description of user's order, encloses it in a plain-text function call and sends to the server via GET request, that can look like this:

192.168.1.201:7777?rpc=XXX

, where XXX is an URI-encoded function call, for example:

Order(”UID1:1;UID4:2;UID23:2;”);

On the server side a local function “Order” is being executed that interprets it an order of 5 items: 1 item of UID1. 2 items of UID4 and 2 items of UID23. The implementation only required a local function and a website-side Javascript code – there was no implementation related to network, sockets, TCP/IP, whatsoever. Later on it was found that the use of prototype implementation helped to solve a problem of resubmitting a html form multiple times (ie. by refreshing a website). Normally solving this type of problem requires using html headers, cookis or hidden form fields with sequential numbers. In the project the problem has been solved by changing the Order function call to new format:

Order(order: string; timestamp: string);

So the same call as given before would now look like:

Order(”UID1:1;UID4:2;UID23:2”,”20130128120000.000abc”);

Where the second argument is a timestamp consisting of the date and time down to a millisecond in the moment of submitting the order with 3 character-long random suffix, just in case there was another order from different user in the same millisecond. Now the user could resubmit the form any number of times without replicating the order. The very first time the function Order is being called with a unique timestamp, it is recorded, and every next time an order with same timestamp arrives, it is handled as a request to view the already-existing order, not to create a new one. Exporting function *Order()* which was already present in the

project's source code prior to the upgrade, took minutes. Some more time had to be spent to create a web interface (HTML page with CSS and Javascript), but the whole upgrade took only 1 working day to deliver, which would not be the case if the API sharing prototype was not used in the development.

The feature of remote control of music playback in the restaurant took approximately 10 minutes to implement, as it only required to export few existing functions to the resource sharing network: *Play(); Pause(); Stop(); Next();* and *Prev();* and an HTML interface took another 20 min to create.

On top of that, since the API Operator integrated into the code of Restaurant's software runs an WWW server, the end user interface is available through a web browser, which makes it a cross-platform solution.

4.4 Study of a prior art

One existing solution that seems to be similar to the proposed one is WOSH Framework (wosh.sourceforge.net), which stands for Wide Open Smart Home. It is also a middleware framework that is Service Oriented (SOA) and enables integration of devices in a network. Additionally WOSH is a free and open source.

4.4.1 Case study

- built-in end-user applications: console and graphical both
- distributed computing, 'zero-configuration' networking (UDP, TCP)
- multi-user, role based access
- remote control using Instant Messaging (using libgloox; compatible with GTalk), SMS (send/receive) and call monitoring (on Windows-Mobile, RNDIS connected smart phone)

- appliances and sensors (X10 devices) monitor/control (on POSIX, based on Heyu)
- entertainment, multi-zone media playback (using GStreamer, MPD on POSIX or QT Phonon on Windows, VLC). Media-Director service provides a high abstraction layer, the recovery/guess multimedia status, hardware/software shortcuts and more
- centralized communication system (selecting best communication channel), using also interpreters (such as Festival for Text2Speech)
- building abstract-representation of the home and its devices (rooms, lights, audio-box, ..)
- Cron and Automation services, providing support for basic every-day tasks
- Weather service, gathering and merging information from various sources

4.4.2 Comparison of main differences

Feature	WOSH	Proposed system
Main programming language used	C++	Pascal
Built-in end-user applications	Yes	Yes[1]
'zero-configuration' networking	Yes	Yes
Multi-user, role based access	Yes	Yes
Remote control using Instant Messaging	Yes	No[2]
Base communication protocol	messages	HTTP, Plain-text over TCP/IP
RPC vs RMI approach	RMI	RPC
Message/RPC-delivery contract	weak[3]	strong[4]
Message/RPC ordering	FIFO, UDP-based	FIFO, TCP-based

[1]Built-in applications include RPC gate and configuration wizard (both via webpage)

[2]the feature is not provided as a ready component, but the architecture supports adding it

[3]no ACKs/retransmissions

[4]RPC ACKs and ACK-error handling via events

Very important part of both systems is the RPC/RMI engine. In the case of WOSH there are stubs and skeletons responsible for RMI. A stub is a proxy for a remote object that runs on client computer. A skeleton is a proxy for a remote object that runs on the server. Stubs forward RMI (along with associated arguments) from clients to skeletons, which forward them to the appropriate server object. Skeletons return the results of server method invocations to clients through stubs.

The design of the proposed system is very different when it comes to addressing RPC/RMI. The primary difference is that WOSH uses RMI, and the proposed system uses RPC. In the RPC there are no objects stored neither on the client nor the server side, which would allow for their methods to be invoked remotely. This is also different from major SOAP/CORBA based solutions. This also allows for exporting functions from the originally local code up to the SOA without the need of designing objects for RMI. On the client side, there is only a generic wrapper function whose only argument is a dynamic length array of strings. The client also incorporates a small translator responsible for unmarshalling incoming RPC encoded as plain text messages over TCP/IP into an array of strings containing text-encoded values for arguments required by the wrapper. The client side trusts the server side, so that no additional control is being done during the unmarshalling process. After the wrapper function executes, the translator is responsible for marshalling its result and the execution status and returning it to the server. On the server side, there is a built-in interpreter, that receives plain-text (or URI encoded text via Query String arguments) and parses it to recognize a valid syntax, with valid reference to a remote function/procedure, and with valid constants values for required arguments, or valid reference to variables of proper type, or valid references to other functions with proper result type. All the references are being resolved by the server prior to sending RPC to

the client side, and all the arguments gets to be passed as constant values at the end. Any invalid RPC call would be recognized as broken by the server-side interpreter before it could be passed on to the remote client. The advantage of such approach is that it is possible to add a code optimizer to the interpreter, so that an entire block of code including RPC calls, can be investigated for dependencies to allow buffering and/or caching of certain data values to limit the number of necessary remote calls in within the scope of a single script. Each individual RPC call is only grouped under a domain, not under an object on the server, which seems to be more transparent. Implementing a similar code optimizer for the RMI approach would require dealing with more complex syntax of the source code by the parser. Ultimately, WOSH lacks the interpreter component in its design

5 Conclusion

Proposal architecture for integration of hardware and software resources turned out to be possible to implement, and was also successfully tested in a commercial project. It has proven to save time and complexity when adding RPC functionality to an existing system.

Some of the directions for the future work are:

- Publish an edition of API Operator and SDK online, to attract potential developers in using it for their projects

- Promote a concept of exporting functionality of commercial programs as an added-value (promoting compatibility with the system)

- Improving the support for EUD by adding supported semantics to the built-in script interpreter, and perhaps by developing a user-friendly source code editor

- Porting API Operator software and SDK to more platforms

There are 2 important aspects that make the proposed system original:

1. The alternative approach
2. Incentives for supporting the philosophy

In terms of the approach, other service-oriented systems allowing for integration of resources and enabling RPC, generally expect developers to be willing to intentionally build solutions meant to either be used as a service, or to incorporate and utilize support for RPC. The approach of the proposed system is that the developer does not have to build solutions with these intentions. In fact, the solution is also dedicated to developers who do not see the need for RPC support (and related) or service-like usage in their code. They agree to make their code compatible with the system only with the consideration to the end-user's convenience, and as an added value. By making the compatibility, they manifest the philosophy that the end user is creative enough, and capable enough to figure out his own ways to use the product, outside of the manner or problem domain anticipated by its developer in the moment of creation. And by providing the system compatibility of their products, developer manifest that end-users deserve that freedom. The developer does not have to especially adapt his programming style or his architecture to show consideration for RPC support or integration into a network as a shared resource. He just allows for an alternative input interface apart of the end user directly interacting with his program. The alternative is to accept input coming from the network via RPC call, which happens transparently to the program itself. Moreover, the developer can even decide to share a code that if called, does not affect the program itself in any way. For example, allowing to perform an operation on an argument, without affecting any of the documents currently being edited by the program.

As for the incentives, when integrating the solution into existing products, both the target for the product and the market penetration potentially increase. The target grows because now also users without interest in the domain, for which the product was originally designed, may

find it useful for their own problem domains, even though they were not anticipated by the developer. The market penetration can potentially grow as well, because some of these end users not anticipated by the developer may actually purchase the product (if it is not freeware). Other benefit for the developer is the ability to implement more sales strategies. For example, his product might be offering a separate licensing terms and pricing for the end users who will only use it as a standalone product, different from those when the end user will also use the product as a resource under the system, and perhaps also different from if the end-users is planning to use the product in both ways. In other words, the incentives for making existing product compatible may come on many levels.

From the design point of view, API Contributor drivers and the RPC interpreter were not found to be realized within alternative solutions. Moreover, during the research the feature of flavors for API Operator and API Contributor editions was also not found in other solutions.

6 Bibliography

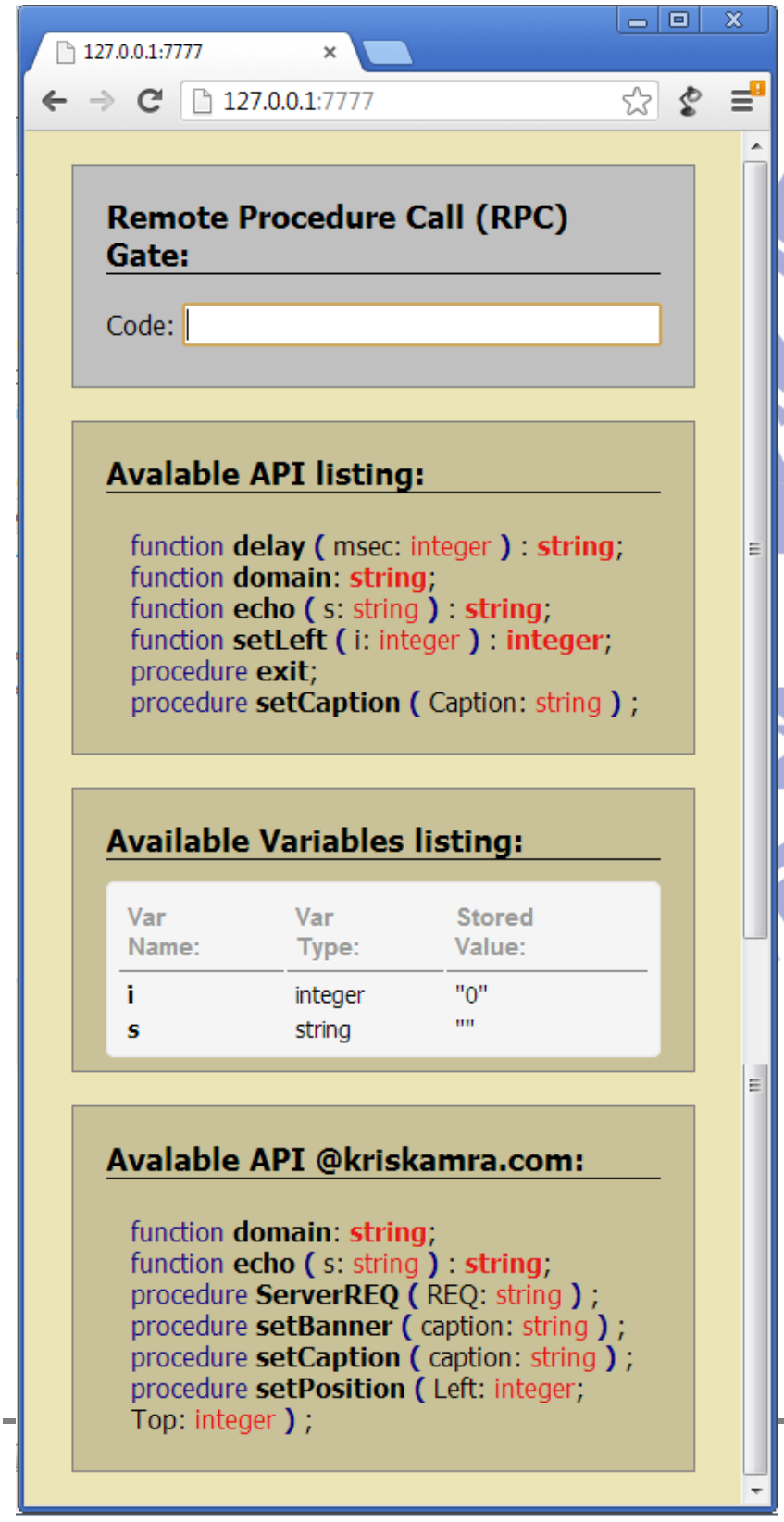
- Scaffidi, Christopher, Shaw, Mary and Myers, Brad A. (2005): Estimating the Numbers of End Users and End User Programmers. In: VL-HCC 2005 - IEEE Symposium on Visual Languages and Human-Centric Computing 21-24 September, 2005, Dallas, TX, USA. pp. 207-214
- Burnett, Margaret M. and Scaffidi, Christopher (2013): End-User Development. In: Soegaard, Mads and Dam, Rikke Friis (eds.). "The Encyclopedia of Human-Computer Interaction, 2nd Ed.". Aarhus, Denmark: The Interaction Design Foundation.
- Lieberman, Henry, Paterno, Fabio, Klann, Markus and Wulf, Volker (2006): End-user development: An emerging paradigm. End User Development. In: Lieberman, Henry, Paterno, Fabio and Wulf, Volker (eds.). "End User Development (Human-Computer Interaction Series)". Springerpp. 1-8
- FRAMINGHAM, Mass. May 24, 2012, IDC Press release: "Android- and iOS-Powered Smartphones Expand Their Share of the Market in the First Quarter, According to IDC", Available online at: http://www.idc.com/getdoc.jsp?containerId=prUS23503312#.UPv0kiesiSo
- Darcy Travlos, “Five Reasons Why Google Android versus Apple iOS Market Share Numbers Don't Matter”, Forbes 2012, available online at: http://www.forbes.com/sites/darcytravlos/2012/08/22/five-reasons-why-google-android-versus-apple-ios-market-share-numbers-dont-matter/
- Arnaud Sahuguet, Fabien Azavant (2001): Building Intelligent Web Applications Using Lightweight Wrappers. In Data and Knowledge Engineering, Volume 36, Issue 3, p. 283-316.

- Mike King, 9 November 2012: Global home automation systems market forecast to reach more than 4 million units by 2013, at http://www.companiesandmarkets.com, News/Information Technology.
- Steven Castle, December 20 2011: We'll Say it Again: Automation is Key to Home Energy Management, at: http://greentechadvocates.com
- Joëlle Coutaz, 2008: End-User Programming for the Home: a Challenge, Proc. @home workshop in conjunction with Pervasive 2008

7 Apendices

7.1 Apendix 1

Appendix 1: Screenshot of a Layer 2 GUI via web browser



Example via local host

Input for RPC calls

Listing of local API

Listing of global variables

Listing of a remote resource

7.2 Apendix 2

Appendix 2: Listing of attached CD

Folder **V1**:	Folder **V2**:
1 libEngine.pas **2 libErrors.pas** **3 libHostetFunctions.pas** **4 libHostedVariables.pas** **5 libSupportedTypes.pas** **6 libTCPserver.pas** **7 NETPASKALv1.exe**	**1 CLIENT** **1.1 RPCclient.dpr** **1.2 RPCclient.exe** **1.3 Unit1.pas** **2 libMemberTCPLite.pas** **3 libServerMod_HTTP** **4 netpaskal.css** **5 Netpaskal32.dpr** **6 Netpaskal32.exe** **7 Tutorial.txt** **8 uAnLex.pas** **9 Unit1.pas**

Description of listed libraries can be found in the chapter 3.7.

There are two folders on the attached CD: V1 and V2. The V1 folder contains the core libraries that were implemented from 2010 to 2011 before the final design for the proposed system was clear. The folder V2 contains additional libraries specific for the proposed system. The API Operator is implemented in the project Netpaskal.dpr and compiled as Netpaskal32.exe, the API Contributor demo is implemented in the subfolder CLIENT in the project RPCclient.dpr and compiled as RPCclient.exe. Both API Operator and APO Contributor projects include V1 libraries.

The "Netpaskal" word originates form an old project that I was working on between 2009 and 2010, that was about implementing a Pascal dialect interpreter as a web service. The subset of Netpaskal project was used to create the core engine for the proposed system, and eventually the name "Netpaskal" remained as a name of some files in the current project.

8 Resume and CV

"The use of traveling is to regulate imagination by reality, and instead of thinking how things may be, to see them as they are." – Samuel Johnson

Personal & Contact Information 个 人 資 料 / 聯 繫 方 式

Name	姓名	Krzysztof Kamil Jacewicz 中文名字: 秦学思
Sex/birth	性别/生日	Male 男 / 28.10.1984
Nationality	國籍	Polish 波蘭
Passport	護照號碼	AP8479012
Current Residence	當今居留	中華民國, Taiwan (R.O.C.), ARC/居留證: JC00811568

Current Address 當今住址:	中華民國 30072 新竹縣新竹市寶山路 13 號 4 樓之 A No.1 Minsheng Rd., Alley 10, 6F/2, 30072 Hsinchu City, Taiwan (R.O.C.)
Mobile 手機號碼/ e-mail:	**+886 989 154 500 /** k.k.jacewicz@gmail.com
Instant Messaging:	**MSN:** misheque@konto.pl, **Skype:** k.k.jacewicz
www resume:	http://www.goldenline.pl/krzysztof-jacewicz http://www.linkedin.com/in/krzysztofjacewicz

Education 教 育 背 景

School: **國立交通大學, National Chiao Tung University (NCTU), Hsinchu, Taiwan** (since 2009-2013)
Major: Electrical Engineering and Computer Science
Level: Graduate Program （Master Degree)

School: **Adam Mickiewicz University (AMU), Poznań, Poland** (2005 - 2006)
Major: Computer Science
Level: Undergraduate Program, o(Bachelor's Degree)

School: **University of Warmia and Mazury (UWM), Olsztyn, Poland** (2003 - 2005)
Major: Computer Science
Level: Undergraduate Program
(Transferred to AMU for the last year of studies and graduation)*

School: **Adam Mickiewicz University (AMU), Poznań, Poland** (since 2005)
Major: Sinology (Studies of China)
Level: Graduate Program
(Susspended on the third year, after one year of remote studies from Beijing, due to job and study opportunity in Taiwan)*

Language Proficiency 語 言 技 能

Polski: Native language
English: Excelent, natural second language
Français: Fluent, certified by DELF - Diplome d'Etudes en Langue Francaise
中文: Mandarin, Fluent, studied as major, supported by BCT cert.(The Business Chinese Test, 商务汉语考试)

Résumé

Education

BA in Computer Science from Poland and MA in EECS from Taiwan both from renowned Universities gave me the engineering background, while additional study in Sinology gave me the background in linguistics and cultural study, with particular focus on Chinese language and culture. I'm fluent in 5 languages: Polish, English, French, Mandarin Chinese and Pascal programming language.

Work Experience

My 5+ years professional work experience since 2007 was always in the field of IT, including 1 year work in the Mainland China (Beijing) and 4+ years in Taiwan (Hsinchu, Taipei). I have been working in a multidisciplinary teams as a "bridge" specialist between RND and business planning.
On the engineering side I have been involved in architectural design of IT systems and SOA, as well as in programming (Delphi, FreePascal, C++, DHTML), including: client-server applications (Windows, Linux), embedded components (ARM-linux, Motorola-AmigaOS), mobile (Android, Java MIDP), server-side binary CGI (Linux, Windows), End-user desktop applications and middle-ware.
On the business-side I have been involved in business planning, creative marketing (incl. Social-media and rich media content), business development, building and maintaining of distribution channel, Sales design (incl. SaaS model) and customer relations building.
Additionally I have been involved in development for automating in-company processes by implementing small-scale software for internal use.

Extracurricular Experience

Photography & filming: folio.kriskamra.com, fb.kriskamra.com
Traveling: http://kris-travels.blogspot.tw/
Martial Arts (particularly Aikido)

Work Experience 工 作 經 驗

Company: **American Megatrends Inc., Taiwan Branch | 美商安邁科技股份有限公司台灣分公司**
Position: Specialist at Business/Marketing Group
Period: 2010/09 – 2012/05
JD: Marketing; Business planning; Cloud Computing; SOA; Digital Signage; HID;

Company: **Gigazone International Co.,Ltd of GIGABYTE Technology Group | 曜嘉科技**
Position: Account Manager
Period: 2010/03 – 2010/09
JD: Int'l sales to distribution channels; Distribution establishment;

Company: **Ruling Digital Inc. | 銳綸數位股份有限公司**
Position: Product Manager
Period: 2008/07 – 2010/01
JD: Web service; desktop applications;

Company: **Vale Internet (Beijing) Technology Co.,Ltd |网乐互联（北京）科技有限公司**
Position: Product Manager
Period: 2008/02 – 2008/08
JD: Web portal development; media streaming;

Company: **Rawspeed (Beijing) Technology Co.,Ltd |溢速(北京)科技有限公司**
Position: Business Development Manager
Period: 2007/10 – 2008/08
JD: CDN (Content Delivery Networks); P2P; Data transfer over network;