PLUG-AND-PRODUCE TECHNOLOGIES REAL-TIME ASPECTS

Service Oriented Architectures for SME Robots and Plug-and-Produce

Klas Nilsson
Department of Computer Science, Lund University, Box 118, SE-22100, Sweden

Matthias Bengel
Robot Systems, Fraunhofer IPA, Nobelstr. 12, DE-70569 Stuttgart, Germany

Keywords: Plug-and-Produce, Plug-and-Play, real-time computing, flexible manufacturing systems, robot.

Abstract: Plug-and-Produce (with the meaning that devices can be plugged in without any manual configurations needed) is an attractive paradigm for manufacturing systems, and in particular for Small and Medium Enterprises (SMEs) that do not have the expertise of system integrators but do need to be able to reconfigure their systems by themselves. One approach for loosely (in terms of timing) coupled devices is that of Service-Oriented Architectures (SOA). As can be understood from developments with multiple robot arm and online operator interactions, future applications will in some cases need real-time guarantees for performing services. That includes both real-time communication and the need to perform services with a predictable timing. A review of available technologies and inherent limitations of distributed computing leads to the conclusion that the standard SOA approach based on process oriented (like for RPCs and web services) calls similar to distributed object orientation will not be practically useful. Instead, a data or state centric approach should be adapted together with one-way message-based communication.

1 INTRODUCTION

The idea of "just plug in and produce" for manufacturing equipment is very attractive. It is inspired by the Plug-and-Play concept for PCs, which has been developed from something less well working for the old ISA-bus PCs running Windows95, into a quite useful end-user support for plugging in USB and other types of devices. There are sometimes real-time requirements on the communication between PCs and their peripherals, and in manufacturing there are real-time requirements on the communication between different devices.

Such devices in manufacturing can be various types of equipment such as robots, advanced sensors and PLCs, and real-time communication is accomplished via field buses or direct wiring, but then with very limited PnP support. Comparing with the simplicity of connecting appliances to a home PC, one should observe that the PC then is a master device that deals with real-time over dedicated communication lines such as fire-wire for a camera.

The real-time problem is not really there, it just works in a reasonable way when sufficient resources are provided (such as USB2 for a memory stick), and when that is not the case the user has to be a bit more patient.

Large enterprises have technical experts that deal with system integration and setup of communication around robots and other types of machines. The future anticipated wide-spread use of robots in SMEs, however, leads to a situation where non-experts (home PC users expecting PnP to work) need to setup and maintain their robot installations. In this situation, the lack of competence is not a problem; it is a reasonable challenge that automation/computer/software engineers should solve. The problem is, however, the lack of awareness within the involved engineering disciplines concerning the inherent or hidden limitations that pop up as peaks of complexity for the end user, which efficiently prevents the PnP

Nilsson K. and Bengel M. (2008).
PLUG-AND-PRODUCE TECHNOLOGIES REAL-TIME ASPECTS - Service Oriented Architectures for SME Robots and Plug-and-Produce.
In *Proceedings of the Fifth International Conference on Informatics in Control, Automation and Robotics*, pages 249-254
DOI: 10.5220/0001504702490254

paradigm to work in practice in SMEs. A certain infrastructure coping with the (then internal) complexity is needed.

Is explicit support for real-time communication part of the needed infrastructure or not? Well, it is typical for infrastructure (phone lines, high-ways, health care, and so on) that:

- Persons (in this case engineers) have different opinions about what need to be provided and what it left to be solved on a case-by-case basis.
- Persons (still engineers, considering real-time and PnP) imagine a cost for additional features, and if the imagined cost is high compared to the imagined (not actual) benefits, there is a resistance of supporting development of the support.
- Lack of a complete supporting infrastructure can result in complete systems being developed in parallel, with deficient overall efficiency, since experienced problems (even if only in few cases) may be severe.
- A well-working infrastructure is not really noticed since it is taken for granted (only noticed when not working), so even persons that actually depend on it may pay no attention to it.

Returning to PCs and IT infrastructure, a comparison with security is relevant. Security should better be built in using solid principles within device drivers, operating systems and in middleware solutions. Since that has not been the case for typical PC systems, the costs in terms of failing systems/enterprises and add-on protective systsems (hardware and software) has been enormous. In development of future automation solutions we should be extra careful since the field by itself is not big enough to cover huge extra costs; instead we have to benefit from low-cost solutions. Therefore, if PCs and low cost devices (hardware and software) do not provide the needed solutions, we should pay extra attention to get the foundations right for future efficient SME usage.

Is then real-time communication and real-time execution something we need; is it something that can be added afterwards; or what should be the foundation for future applications? The standard answer today is that real-time is needed in very few cases, so let's neglect it and focus on other application aspects.

One promising approach is then so called Service-Oriented Architectures (SOA), which could be quite suitable (at least for non-real-time parts) but the implementations tend to be slow and hard to map onto real-time suitable implementations. For instance, both UPnP and web services are implemented on top of http with XML-based information structures that not necessarily map on hardware supported real-time means of communication. We will come back to this issue, but first some preliminaries that our discussion can benefit from.

2 PRELIMINARIES

So called middleware and models of distributed components providing software services typically come from enterprise systems, but is also being used for mobile robots and other system interconnections with no strict real-time requirements. For real-time communication within industrial automation, the current practice is based on field buses. In modern integrated systems with the need for so called vertical integration, there enterprise and device levels need to be able to communicate and the different technologies need to be unified.

2.1 Basic Model of Communication

Software developers today normally use object-oriented programming, and from the beginning they learn how to use method calls for object interaction. Multi-threaded applications, today typically written in Java or C# with language support for synchronized methods, also follow the object-oriented paradigm quite well, which means that two-way synchronous communication is the basis for inter-object communication within a single program.

Developers with experience from computer networking or from data-flow oriented applications with needs for buffering of asynchronous messages may build distributed applications differently than local programs, simply to deal with the quite different and complex communication reality. However, along the lines of hiding complexity, a perhaps more common trend is to stretch the object-oriented paradigm to cover also distributed systems, which by definition are concurrent (but so far we assume no real-time requirements). This is also the basis for several of the middleware approaches that are listed below. The problem is, however, the distributed object-oriented paradigm has limited applicability when assumptions about the networked object interaction do not hold. That is, for realistic applications (such as robot work cells) the networking for object interaction does not scale or does not handle typical deployment contexts. This has been known for over a decade as appropriately described by (Waldo, 1994), and the reader is suggested to read at least sections 2 and 7 of that report before continuing here.

In our case dealing with automation and real-time demands (not to mention safety), the situation is even more difficult, but still some of the dead-end approaches are being promoted. To review the current situation some further details on existing solutions and requirements now follow.

2.2 Fieldbuses

The classical field buses are usually setup in a ring-like structure, which is natural since the original purpose was to minimize cabling. Some of them provide hard real-time capabilities (like Sercos), others just implement soft real-time like Profibus and CAN bus. There, messages can almost be made sure to be delivered in time – depending in their priority. Nevertheless, the CAN bus supports distributed hard real-like control although the communication itself doesn't (CAN, 1991). These field buses make use of their own physical layer. Sercos uses fibre optics, CAN bus relies on a three-wire cable.

Typically a rather limited number of devices can be connected to one network. Profibus and CAN bus for instance support up to 127 participants.

Newer developments don't use their own hardware layer but rely on the Ethernet technology. Well-known examples are Sercos III (Sercos, 2002), ProfiNet (Profibus, 1999) (ProfiNet is part of the Profibus specification since 2003) and EtherCAT (ETG, 2007]. All of them implement real-time capabilities. This is mainly done by replacing some of the ISO/OSI layers (Zimmermann, 1980) in the standard TCP/IP stack. Therefore, the wiring gets less complex and requires less effort. Further more, in some of these technologies – if hard real-time is not required – standard hardware like network switches from the office world can be used.

As these technologies make use of the IP technologies, the number of participants gets larger and is not limited to only a small number. One fairly successful attempt to bring together several protocols under a unifying overall framework is the CIP (ODVA, 2006) initiative. However, such general solutions go with lengthy descriptions and detailed APIs that are not easily adopted.

2.3 Middleware

Apart from the field bus technologies there are several approaches to communicate not only between different automated hardware devices but as well between different programs, spread in the network. Currently, some main directions can be observed:

2.3.1 Web Services

The web services are a mainstream technology in the B2B (business to business) communication and are used there mainly as Enterprise Java Beans (Sun, 2007) and Microsoft's .Net technologies. The web services are usually implemented using the SOAP protocol (W3C, 2007) that communicates via Http and TCP/IP over Ethernet. The main advantage is its flexibility and availability in the intranets and the internet due to using the http port. But in this scope, the main disadvantage is the lack of real-time capability due to the standard network protocols.

2.3.2 OPC/UA

OPC/UA (OPC Unified Architecture) is the newest of all OPC specifications (OPC, 2007). It contains an own communication stack which is scalable from embedded controllers up to main frames.

The architecture follows the SOA paradigm (service oriented architecture) including several logical layers. It supports profiles which can be queried. Therefore, communication partners can query their provided services.

As OPC/UA was invented for communication via the internet, its architecture supports security features like encryption and authorisation. Determinism is not included. Internal tests discovered that round-trip times are short enough to implement even control loops for devices in automation technology.

2.3.3 Corba

Particularly in the research area different Corba implementations are used widely. Corba is the abbreviation for Common Object Request Broker Architecture and is a specification of an object-oriented middleware (Mowbray, 1997). Its core is a so-called object broker which defines platform-independent protocols and services.

Usually the program code for the communication over the network is not written manually. Instead, an abstract language, the Interface Definition Language (IDL) is used. From that, the stubs and skeletons are generated automatically for various programming languages and different operating systems. This is why Corba is platform-independent on the one hand and programming language-independent on the other hand.

Newer implementations of Corba even support real-time, provided that the underlying operating systems and communication channels do as well.
Due to the fact that there are several different Corba implementations it cannot be assumed a priori that all the different implementations interoperate well as

there seem to be some differences in the concrete implementations. The specifications for Corba and IDL can be retrieved from (OMG, 2007).

2.3.4 Universal Plug and Play

The main purpose of Universal Plug and Play (UPnP) is to control devices independent of their manufacturers. UPnP is well-known in controlling routers and multimedia equipment.

Originally, UPnP was introduced by Microsoft, but nowadays, certifications for devices are performed by the UPnP Forum (UPNP, 2007) which at the time of writing consists of 845 vendors.

UPnP can be used on any communication channel supporting IP communication. Basically, UPnP makes use of several protocols for discovery, addressing, description, eventing and so on. Also the technologies IP, UDP, Multicasting, Http, and SOAP are well known and are used in this technology.

Unlike some the other middleware standards described here, UPnP does not support any security features. A good introduction to UPnP can be found in (Jeronimo, 2003).

2.3.5 Representational State Transfer

The standard way of implementing web service (using SOAP as in UPnP) has a number of drawbacks in terms of (Newmarch, 2005):

1. Inefficiency with XML-based RPC-like communication on top of http.
2. Unclear semantics in the use of GET and POST requests.
3. Unclear object model and deficient referencing of attributes in nested data structures.

To overcome these difficulties, the REpresentational State Transfer model was suggested by Fielding (Fielding, 2000) to overcome the above drawbacks.

Technologies used in Microsoft Robotics Studio® (MSRS, see Microsoft.com for latest info) is claimed to include a lightweight REST-style service-oriented runtime, but the Decentralized Software Services Protocol (DSSP) is actually SOAP based. Therefore, even if DSSP is oriented towards exposing device states, it is not clear how the transfer of state information can be mapped to real-time eventing as we aim for.

3 FUTURE COMMUNICATION AND MIDDLEWARE

For PnP automation devices to become a reality, it must be easy and streamlined to develop such devices. The reason is that the strong arguments for interoperability as in telecommunications do not apply to automation, which also is to small an area to cover extensive developments of special solutions. Hence, we must be able to make use of available technologies in a swift manner.

Note that the solution is not standardisation, at least not in the traditional sense with agreements that are negotiated in committees and then maintained as thick documents. There are already a lot of standards; we do not need more of them (unless there is a core new technical solution that calls for some agreements on how to make use of it).

What is the suitable approach then, and what are the requirements?

3.1 Requirements for R&A Systems

To support the desired PnP developments we need middleware providing an API with a suitable expressiveness and simplicity. These two demands are contradictory and it is an open issue if a good solution can be found or not. For instance, even if there are abstractions and APIs for communication channels, the profiles and specifics for fieldbuses may add too much of complexity if the selected abstractions do not map onto the actual setup. Another example is CORBA that should simplify programming of distributed applications, but still adds too much complexity (for programming, for deployment, and for troubleshooting). Thus, abstractions need to be defined on an appropriate (probably medium) level to avoid problems:

1. Too low level: The complexity of networking and field-buses gets exposed and the API gets useless for a majority of the developers.
2. Too high level: Communication setup not reflecting actual needs in terms of timing and resources, and a variety of APIs will evolve.

Hence, the middleware should deal with communication in terms of an application oriented (not networking oriented) API that is designed to map well on the most suitable alternatives for real-time communication. To structure the topic we may separate between:

- Mechanism: How are things accomplished technically, typically locally in a computer node or on the network?
- Policy: How are the mechanisms used and configured locally?
- Deployment: How is an actual system configured before and during run time?

Taking a look at just the mechanisms for communication and connections between devices, there are at least the following to consider, some if it

affecting the software in a less obvious way. For brevity, the following items (that each deserves one paper) are very shortly described without references:

a) **Driven by time or events**: EtherCAT, TTP, and RTNet are time driven while ThrottleNet and normal socket connections naturally forms event driven communication. Combinations?
b) **Exposing services or data**: Configuration methods and operational state, RPC or REST?
c) **Operating peer-to-peer or master-slave**: Is an EtherCAT slave PC node a slave or a peer?
d) **Connecting peer-to-peer or client-server**: Server socket in server, client, or p2p software?
e) **Topology as star or ring**: Implications on predictability, reliability, resources and cabling?
f) **Connection-based or datagram channels**: How to deal with the tradeoff between performance and reliability/simplicity?
g) **Synchronous or asynchronous**: Both events and calls can be both. Best practices and APIs?
h) **Bidirectional or one-way**: Should there be a built-in support for handling event replies, for unreliable low-cost means of communication?
i) **Hot-plugging or reset**: EtherCAT connected to an end-effecter via tool exchanger, is a communication dip during tool changing acceptable?
j) **Predictable or best-effort**: Specification of performance requested or obtained, but what does it mean for the application software?
k) **Dependable or fail-safe**: SME robots only need to be failsafe or are there mission-critical cases?
l) **Guarded or collaborating nodes**: Does human-robot space-sharing imply a need for the 'babbling-idiot' protection as in TTP.

Since not all this complexity should be exposed in a complete API (that nobody would use), we need:

- Tradeoffs such that the most critical and common cases are well supported, for instance by suitable default configurations.
- Abstractions in layers and a guide such that only a few types of configurations are needed in actual scenarios. Integration with model-driven design tools is desirable.
- Ontology-based definitions of the communication model, including formal definitions of items a to l above. Today standards and definitions are only expressed in documents (for humans) and code (for computers), but there is no meaning including semantics that is understandable by both humans and machines, which is necessary for application-level PnP.
- Open source reference implementation working with some generic devices. Different vendors will then adopt the software (or perhaps re-implement in other languages and for other platforms) but agreements and specifications need to be with respect to actual runnable code.

Suitable tradeoffs with respect to embedded distributed software for robot work-cell devices should primarily suit low-cost solutions as needed for SMEs. Our research indicates that the following decisions are appropriate:

A. Real-time data-flows should be based on one-way data streams that from a programming point of view is equal to an event or message stream, but only resource use should be defined programmatically (e.g. by providing a maximum size message and a maximum frequency) and no configuration of communication profiles should go into the application code.
B. There needs to be a binary version of the real-time data flows, with complete description of message types when a connection is established but with only minimal binary information during real-time operation. That way most control messages fit into Ethernet frames and low cost raw Ethernet can be used for predictable communication.
C. Real-time RPC, RMI, CORBA method calls, or web services, should not be permitted, at least not the standard IDL way. If permitted, the underlying asynchronous operation should be explicit, meaning that there is a call object that can be queried for completion, errors, etc.
D. Non-real-time network traffic should be possible to do in the same way as for real-time communication, but in this case synchronous method calls could map (automatically) to RPC calls or web services.
E. All encapsulated entities used by the real-time parts of the application should be resource aware; real-time is just a special case of resource limitations (namely CPU power and the scheduling of it), so also memory usage and IO bandwidth allocation should be taken care of in a structured way.
F. The use of safe languages such as Java and C# should be used for improved modularity and robustness of hand-written code. Unsafe languages such as C++ should not be used for flexible dynamic parts of systems since the risk for dangling pointers and crashes get too high.
G. All systems should (without extra engineering) run on standard desktop computers for simulation and debugging purposes, then without real-time performance but with full concurrency using a virtual time scale.

There will of course be no power to enforce the decisions above and standardization via a committee would not work; freely available implementations of selected abstraction must be the most simple and efficient way of building systems, and thereafter de-

facto standards should evolve. In this perspective compatibility (either directly or via bridges) with major wide-spread middleware solutions such as MSRS must be supported. A variety of initiatives are ongoing, including the Apache CXF Open Source Service Framework (Apache, 2007).

The focus in REST on data rather than methods (or nouns instead of verbs) suits our manufacturing scenario quite well since it is data that is actually transferred over the network and simple mappings of device state to network data should permit tiny devices to be part of the PnP system. Using switched raw Ethernet (Martinsson, 2002) and self-descriptive data packets (Blomdell, 2007) then supports low-cost solutions. To find out if the above technical decisions are appropriate or not, more assessments are needed to get application experiences. That is ongoing but outside the scope of this paper.

4 CONCLUSIONS

Appropriately designed real-time capable middleware and PnP support will most likely simplify for application development rather than being a complication. Support for real-time communication should be built into the abstractions we use for communication between programs and computers. Real-time support means permitting real-time operation (when OS and all involved parts comply), so well-written applications will provide real-time capabilities when deployed on a real-time capable system. Many promising technologies and partial solutions have been developed over the years, but it appears there are no solution with the completeness and scalability that is needed for the future very flexible and modular SME applications. A suitable approach appears to be open-source reference implementations of suitable abstractions for Ethernet-based communication and development of middleware that is compatible with (but also useful independently of) the Microsoft Robotics Studio. Additionally, special attention should be paid to self-descriptive binary communication channels that map well onto raw Ethernet and that can be bridged automatically to high-level XML-based eventing as used in several of the existing standards. Such developments are currently ongoing in the SMErobot consortium.

ACKNOWLEDGEMENTS

This work was supported by the EU FP6 project SMErobot®.

REFERENCES

Apache, 2007: Apache CXF: An Open Source Service Framework, http://incubator.apache.org/cxf

Blomdell, A., 2007:The LabComm Protocol Language, http://torvalds.cs.lth.se/moin/LabComm

CAN, 1991: CAN Specification, Version 2.0, ISO 11898: 1993-11

ETG, 2007: EtherCAT Technology Group website, http://www.ethercat.org

Fielding, R., 2000: Architectural Styles and the Design of Network-based Software Architectures, http:// www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Jeronimo, M., 2003: UPnP Design by Example. 2003 Intel Press. ISBN 0971786119.

Martinsson, A., 2002: Scheduling of Real-time Traffic in a Switched Ethernet Network, Master thesis, https:// www.control.lth.se/database/publications/article.pike?action=fulltext&artkey=5683

Mowbray, T. J. and Ruh, W. A., 1997: Inside Corba: Distributed Object Standards and Applications. Addison Wesley 1997. ISBN 978-0201895407.

Newmarch, J., 2005: A REST Approach: Clean UPnP without SOAP, Newmarch, J., "A RESTful approach: clean UPnP without SOAP," *Consumer Communications and Networking Conference, 2005. CCNC. 2005 Second IEEE* , vol., no., pp. 134-138.

ODVA, 2006: The Common Industrial Protocol (CIP™) and the Family of CIP Networks. At *http://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00123R0_Common%20Industrial_Protocol_and_Family_of_CIP_Netw.pdf*

OMG, 2007: OMG Specifications. – Middleware Specifications.http://www.omg.org/technology/ documents/spec_catalog.htm

OPC, 2007: The OPC Foundation. http://www.opcfoundation.org/Downloads.aspx?CM=1&CN=KEY&CI=283

Profibus, 1999: Profibus Specification. 1991/1993 DIN 19245, IEC 61158/IEC 61784

Sercos, 2002: SERCOS interface, 2002. IEC/EN 61491.

Sun, 2007: Enterprise Java Beans, Sun Microsystems, 2007. http://java.sun.com/ejb

UPNP, 2007: The UPnP Forum. http://www.upnp.org/

W3C, 2007: SOAP Specifications, 2007. http://www.w3.org/TR/SOAP/

Waldo, J., Wyant, G., Wollrath, A., Kendall, S., 1994: A Note on Distributed Computing. http:// research.sun.com/techrep/1994/smli_tr-94-29.pdf

Zimmermann, H., 1980: OSI Reference Model — The ISO Model of Architecture for Open Systems Interconnection, IEEE Transactions on Communications, vol. 28, no. 4, pp. 425 - 432.