Human Aspects of Machine-to-Machine Communications and Cooperation

Jaroslav Král¹,² and Michal Žemlička¹

¹ Dept. of Software Engineering, Charles University in Prague / Malostranské nám. 25, Praha, Czech Republic / {kral, zemlicka}@k5i.mff.cuni.cz
² Dept. of Computer Systems and Communication, Masaryk University Brno / Botanická 68a, Brno, Czech Republic / kral@fi.muni.cz

* Corresponding Author: Michal Žemlička

Received September 28, 2011; revised November 9, 2011; accepted December 10, 2011; published December 31, 2011

Abstract: The growing power of modern IT enables the growing autonomy of individual computer systems as well as growing autonomy of the subsystems enabling their communication and collaboration. Such systems are continuously growing, tend to be more open, and tend to communicate with other systems. In this paper, we show that they can be seamlessly implemented if the machine-to-machine communication systems enable an easy human involvement into the agile development, (agile) supervision, and (agile) use of such systems. A feasible solution to the latter issues can be quite easily achieved if simple modifications are made to the attributes of certain service-oriented systems. We show that machine-to-machine communication systems can and often should be designed as component systems using connectors as services. The networks of such connectors behave like a service providing an architecture. The resulting systems are very flexible, applicable in large service-oriented systems as well as in modern computer networks. It is an important engineering advantage. The connectors as services are very useful in large information systems having service-oriented architecture. They increase flexibility, and thus enable agile development methods of such systems as well as the business processes they support. The existing opportunities have not yet been fully explored.

Keywords: Architectural Service, Architectural Component, User-oriented Machine-to-Machine Communication, System Openness

This research was supported by a research grant from the Grant Agency of Czech Republic [GA201/09/0983, Agile Systems and Service-Oriented Software].

DOI: 10.6029/smartcr.2011.02.002
Introduction

Communication among computer systems is common. The resulting machine-to-machine communication (M2M) system can, in principle, work without any human involvement.

It is false to conclude that the communication protocols, especially message formats, in such systems need not take the needs, knowledge, and abilities of humans into account; moreover, it is false to conclude that the properties of communication should be hardware, software, and developer oriented and need not be user oriented.

It would be true if the quality (desired properties) of the resulting systems was dependent only on these three factors. It is, however, rather an exception than a rule as it in fact assumes that:

• the system requirements can be well formulated at once,
• they do not vary, and
• humans (users) need not understand/analyze communication traffic inside machine-to-machine communication systems.

It in turn implicitly assumes that the system environment does not change.

The assumptions are not true as machine-to-machine communication systems must be open as a rule. The systems must change to meet changing business requirements and technical opportunities. They tend to collaborate with other existing systems and with humans. The analyses of systems failures must start with the analysis of system communication traffic. The traffic must be user oriented (understandable to users) as the users must take part in the analysis of traffic in the case of failures or business contradictions. Therefore, software systems almost always incorporate human aspects. They also influence business processes and have social consequences. We say that such systems involve human interaction. The development of human interactive systems depends on user knowledge domain. It is therefore preferable to design human interactive machine-to-machine communication systems as component systems [1-3]. The capabilities of the components depend on user needs and aims. The interfaces of the components of human interactive systems should be user-oriented. It is especially needed in the systems used by small-to-medium enterprises. User-oriented interfaces have many advantages. They tend to be coarse grained and declarative. They have many user oriented as well as the technical advantages discussed below. The most important advantage is that those systems enhance the capabilities and applicability of service oriented systems. Our proposed systems are very useful for small enterprises. The design of intersystem user-oriented interfaces must be based on the applications of the principles of man-machine cooperation and interaction [4].

Towards Flexible Machine-to-Machine Communication Systems

The size of M2M systems, comprised of networks of hardware as well as software entities, is becoming larger and larger. The systems tend to be open, growing, changing, and dynamically collaborating with other systems and with users.

For some technologically oriented systems, like avionics, these aspects are less visible. They are, however, substantial for human interactive systems. The majority of M2M systems are human interactive systems.

During the gradual development of such M2M systems, there are components (machines) that have not yet been implemented. They must be simulated. One of the most effective ways of solving this issue is based on the following principles [5, 6]:

• Communication partner P of a machine (component) M can be changed to be a portal P0, i.e., the messages generated by M for P are redirected to P0.
• P0 is a portal used as a service.
• P0 can simulate the communication of P with M (Figure 1) answering the messages from M. P0 in fact implements a mock-up prototype. It simulates the communication by a terminal.
• The messages are user oriented, so the communication is user oriented (understandable to users). It implies that the communication is coarse-grained.
This solution is especially simple if the communication is based on the asynchronous exchange of coarse-grained user-oriented messages, e.g., based on extensible markup language (XML). It usually requires almost no additional coding.

User oriented communication has substantial advantages during system use – especially in the cases when M has failed or there have been some other incidents like system misuse. The communication should (must) be analyzed by the users themselves as the needed analyses must be based on deep understanding of the user knowledge domain and user (business) skills, needs, and requirements.

If M1 has no user-oriented interface, then we can use (service) adapters (wrappers) transforming sequences of messages into user-oriented coarse-grained ones, and vice versa (see Figure 2 or [6]). We call such adapters according to [7] front-end gates. Front-end gate, in the case when the system uses asynchronous communication, is in face a service adapter used as a service. It is important to note that front-end gates are technically standard components in machine-to-machine communication systems. Front-end gates can be instructed by an administrator to change the destination of some messages to a portal.

These principles are easily implementable in the case when the M2M system uses asynchronous message-based communication. The wrapper – front-end gate – enables the efficient implementation of information hiding [8] and meets the requirements of [9]. M1 and front-end gate can be on different machines as well as on the same one. Front-end gates can be integrated with M if appropriate.

The use of front-end gates structures the interface of components into two tiers: the low-level one (provided by the authors) of M1 enables full access to the functionality of M1, and the upper-level one that converts the interface to be high-level and allows customization of the interface for specific groups of cooperating partners. As there can be multiple groups of communication partners having different requirements of the interface of M1, it can be reasonable to create different front-end gates for each such group.
There are several solutions to the problem of changing the destinations of output messages of a front-end gate. The most flexible one is to influence the behavior of a front-end gate by a supervisor (administrator) having access to it. The supervisor access is not shown in Figure 2.

As the low-level interface is provided by the author/owner of the component M, it can be, and usually is, fixed during the development of the machine-to-machine communication system, as M₁ is usually integrated as a black box. Developers are then usually unable to change the interface of M₁. On the other hand, the interface can depend on the changes of the implementation of M₁. The proper application of front-end gates is a flexible and powerful solution of the issues caused by the changes.

The principle of two-tier wrapping with a slightly different purpose is discussed in [10].

**Architectural Components**

Special services providing users with system interfaces are called portals. They hide the system structure. It is reasonable to build multiple portals for a single system - each portal is then dedicated to a special group of users. Typical examples are intranet and extranet access or specific portals for specific expert users (process owners) – see Figure 3.

![Figure 3. System equipped with multiple portals](image)

Front-end gate can be well used during the system maintenance as it enables a user-oriented intelligent logging, and it simplifies the extensions of machine-to-machine communication systems and their integration into (communication with) other systems. The proper use of front-end gates can significantly simplify the implementation of highly open machine-to-machine communication systems. Such systems can be easily integrated into service-oriented systems [7]. There are further opportunities: Front-end gates can be easily modified to act as heads of composite subsystems consisting of a group of components [5], as is shown in Figure 4.

![Figure 4. Head of composite component (HCC)](image)
Any member M of the group is allowed to communicate only with other members of the group and possibly with a head of composite component. Partner components are exclusively allowed to communicate with the group of M via the head of composite component. A straightforward generalization is to allow any component type (i.e., architectural services and basic components providing basic capabilities) to be the member of the group. It is obvious that we can build a hierarchic system of services without any additional coding. It suffices to change the rules of how to use architectural services. This aspect is crucial to achieve extreme system flexibility, thus allowing asynchronous message passing and using architectural components provided that the components are autonomous and loosely related. Such components are services according to W3 Consortium and Object Management Group (OMG). The systems with architectural components (services) have therefore a specific (confederated) service-oriented architecture. In agreement with [6], we call all the generalizations of front-end gates architectural services.

The architectural services can be easily modified to provide very different capabilities: a head of composite service can be a software entity on some hosting machine, or it can be implemented using a dedicated intelligent hardware (router, gateway). A front-end gate having a store of messages can implement various message communication protocols like publish-subscribe. A portal equipped with a business process control data can be used to define and control business processes.

Middleware Issues

The proposed solutions depend on the efficient implementation of basic capabilities of middleware. The capabilities should include something like an asynchronous message passing capability. It is not too strong assumption for current software systems as Internet and many other messaging systems support it and at least some applications using them are based on asynchronous communications. We can use many other message passing tools.

According to our practical experience, there can be problems caused by the fact that some straightforward effective solutions can be excluded by the requirement to use some (obsolete) standards or some low-level component architectures [2].

A frequent source of the problems at the side of developers is their resistance to leave the object-oriented philosophy and to properly use asynchronous message passing coarse-grained communication and to apply the principles of service-oriented philosophy. Developers often prefer remote procedure call that is primitive in synchronous communication environment. Remote procedure call is, however, not user-friendly. On the other hand, remote procedure call enables the development of various formal models for defining the properties of the interfaces (connectors) in component systems [1].

Asynchronous message passing capabilities can be implemented in different ways – even in the framework of a single federated service-oriented system (system consisting of loosely coupled autonomous parts being aware of their communication/cooperation partners that typically need not be searched for; [12]). Some components can use specific communication tools and protocols, if necessary.

Contemporary hardware is powerful enough to enable such a solution, unless very short response times are possibly required. Such cases are now quite rare and can be hidden in encapsulated small subsystems and may be solved in an ad-hoc style.

Message Store

The capabilities of middleware can be enhanced by a specific connector – Message Store. It is an architecture component (service) C that transforms the sequences of input messages of C into sequences of output messages sent to destination components. We can, for example, implement various complex protocols starting from the point-to-point ones. The more complex ones are useful for avoiding the point-to-point antipattern [11].

The rules of determining the destinations can be quite complex. They can be specified using complex algorithms, if necessary – e.g., publish-subscribe.

The destinations can be chosen using parsing algorithms applied to the messages. The most straightforward use of Message Store is to implement capabilities similar to the capabilities of broker or router. The Message Store component consists of (see Figure 5):

- Head of message store (component logic),
- Store of messages (optional),
- Log store (optional),
- Supervisor access (optional)
• Input and output ports.

It is the most complex structure of architecture services. All its elements, especially the supervisor, access and inner store, can be used in all architectural component types.

A general architecture component consists of:
1) component logic (head),
2) input ports,
3) output ports,
4) optional supervisor access,
5) optional log store,
6) optional message store.

It is important that the component is used to purely facilitate the collaboration of other services and developed with a proper intention.

![Diagram of a component logic with input messages and partner services](image)

**Figure 5. Message store**

**Convergence of Machine-to-Machine Communications and Service-Oriented Systems**

Machine-to-machine communication systems with architectural services contain components of two types:
1) basic components (machines) providing basic capabilities and
2) architectural services coordinating other components (the architectural ones inclusive).

Machines communicate via a network of architecture of communication services (Figure 11). It is good to design such a network where front-end gates behave like ports of the network.

Such a structure is similar to the structure of contemporary computer networks – compare computers and routers and the concept of component architecture with connectors as services [5]. Such architectures are also useful in the cases when large dynamic service-oriented systems are to be developed. It is a preferable property as it enables a smooth transfer of development methods and turns from the physical level of communication to the higher levels of communication needed in small-to-medium machine-to-machine communication systems, as well as in large systems having a generalized service-oriented architecture. It is a substantial engineering advantage as it enables the application of similar methods and techniques to the development of very different software systems.

We see that complex information systems having service oriented architecture can benefit from the philosophy of machine-to-machine communication systems if they use architecture services that are in fact connectors, as services. On the
other hand, machine-to-machine communication system designers and developers can and should apply the principles and philosophies of component and service-oriented architectures. It moreover opens the ways of smooth integration of legacy and third-party systems.

**Agile Business Processes**

The tools used in implementing architectural components can be used in the development of components called Process Managers that are used as the definition and control engines of agile business processes. It is not excluded that Process Managers use incomplete or informal specifications of business processes. It is possible to enhance the business process specification in an agile way.

![Diagram of business processes](image)

**Figure 6. Support of business processes**

The implementation of a business process \( P \) can be based on the following principles:

1) Process administrator generates a new instantiation of the component "Process Manager" PM.
2) Process data \( PD \) of PM are computed using a corresponding process model read from process model \( M \) and can be edited on the fly.
3) The process parameters are provided and stored in PM.
4) \( PD \) is used by the process owner to control the process enactment, i.e., to generate sequences (networks) of commands for the components able to activate (commands) the sub-processes or steps of the process \( P \) according the business process requirements. The owner can supervise and change the activations in an agile way. It enables reaction to the changes in business environment or to solve various business issues. It is not excluded that no or incomplete model is used. It is possible that \( P \) is defined in an informal way by plain text.
5) The run (experience with) the process is archived.
6) The experience collected in the archive and logs can be used to:
   a. enhance, possibly using by modeler, the process models;
   b. collect and compute information (e.g., statistics) contributing to business intelligence.

The entire system can start, in principle, with no models or with process models defined in plain text. More sophisticated tools like Business Process Execution Language (BPEL) can be used immediately or later. Our proposal enables the gradual development of the store of models and of business intelligence.

**Real-Time Control Systems**
Simple control systems have the structure (Figure 7) consisting of:

- control logic CL,
- drivers communicating with the controlled systems,
- controlled systems.

![Figure 7. Structure of a simple control system](image)

It is usually possible to design the system as a (virtual) two component structure where the components are wrapped and the wrappers have the capabilities of front-end gates (Figure 8).

![Figure 8. Wrapped drivers](image)

Such a structure, provided that needed responses are not too short, can well monitor the sequences of messages. It is useful during system use.

During system development, the controlled system is first simulated by a hardware prototype to test the drivers and to test the response times of control logic [12]. The resulting situation is shown in Figure 9.

![Figure 9. Hardware simulator](image)

It is possible to decompose CL in the manner described above so that CL has component architecture and to retain the main feature of the system from Figure 9. It is also possible to replace the hardware simulator by a software simulator in the way described above. It has the drawback that in such a case we must use sophisticated tools for proper estimation of response times.

It is often possible to provide functions in the controlled system requiring short response times by a specific component (Figure 10).
**Flexible development process**

Architectural components in confederated service-oriented systems are typically quite small software artifacts that are not difficult to be developed from scratch; i.e., they can be used as white boxes implementing only an integration tool of other components. They can be moreover integrated "by" other architecture components.

The basic components are as a rule used as black boxes as they can be:

- third-party products,
- poorly documented legacy systems,
- the systems of business partners,
- developed as an autonomous body by an autonomous team (This feature is advantageous for multi-team work and for highly structured teams.).

The development is in fact rather incremental than iterative. The integration of the components tends to be the bottom-up one. The flexible prototyping described above along with the possibility to use the communication between architectural connectors enables various development process types, the top-down ones inclusive.

Architectural services offer many capabilities:

- user-oriented interfaces,
- redirection or choice of destinations of output messages,
- rules how given a component is used (see the example),
- algorithms of message transformations.

It enables application of principles of agility [13] in the development process of service-oriented systems.

**Logging and Maintenance of Confederated Service-Oriented Systems**

Communication between the components of a confederated service-oriented system must be during the use and maintenance of S logged. It is needed for the following reasons:

1) The archive can be used to control business processes (correctness, timing, and human involvement) to avoid business failures or to avoid anomalies.

2) The archive can be used to optimize and enhance business processes and business intelligence.

3) The logs can be used in business trials.

It is preferable to use the message coarse-grained and user knowledge domain oriented.

It is a simple task to equip architectural component C by a memory storing (selected) messages (together with time stamps) coming to or leaving C. C can then have (i) input ports for input messages, (ii) output ports for output messages, (iii) port for communication with supervisor, and (iv) logging memory. Different ports can use different middleware tools.

Complex rules can be used where output messages should be sent.
**Powerful Maintenance**

Architectural components and the broad use of user-oriented interfaces are advantageous for system maintenance. Almost all the issues in information systems supporting business, e-government, and social activities must be solved together with the users of the systems. The problems are simplified if communication is based on user knowledge domain. It is very advantageous if court trials must take place.

The redirection of messages can be used for replacing a failed or still missing component by its screen prototype. The failed component can be later replaced by another third-party component or by an enhanced component.

Corrective maintenance (debugging the errors not detected by tests) can and should use main tools and methods used during the development.

In confederated service-oriented systems, enhancive maintenance (new or modified capabilities) is technically a very simple task as the architecture makes it possible to continue the development in parallel or as a part of maintenance. Such an attitude is necessary for very large continuously working systems like e-government, management information systems of global enterprises, health systems, etc.

Adaptive maintenance (transfer on a new platform) is to a high degree reduced/simplified as the transfer is hidden behind the used middleware and communication tools.

**Conclusions**

The development of machine-to-machine communication systems must sometimes use the tools specific for lower levels of communication. This fact is, we believe, overemphasized especially in the cases of soft real-time systems (like the system of mass services type) must be developed. Surprisingly enough, we can in many machine-to-machine communication systems apply turns being quite similar to the turns used in large service-oriented systems. The leading idea is to implement connectors introduced by component architectures as services. The opportunities enabled by such a solution were used and verified in several control systems implemented by the authors of this paper.

Their limitations and potential will be the topic of further research. The systems with architectural components (services) consist of a set of basic machines (components) interconnected by a network of architecture services (Figure 11 – structure of a system with architecture components) via "ports” implemented as front-end gates. The network of architectural services behaves like a service providing architecture as a service.

![Figure 11. Architecture as a Service (AaaS)](image)

It is preferable to design and implement machine-to-machine communication systems so that they use integration tools similar to the tools and attitudes used for large-scale information systems using service-oriented architectures, for example. We attempted to show that it is a meaningful aim at least in the cases when the required response times are not too short.

Note that it has a lot in common with the design and use of the network hardware of modern computer networks as well as with open agile service-oriented systems developed using agile development processes.
We have proposed a new and very powerful service oriented design pattern (compare it with [14,15]) and a very useful clone of service oriented architecture. It seems to be possible that our attitude could be useful in the design of operating systems for massively parallel chip architectures (chips with a very large number of kernels) working in multiple instructions – multiple data (MIMD) mode.

The proper choice of the types and structures of communication protocols as well as the proper choice of architectural services remains to be analyzed. It will be one of the topics of further research and applications in practice.

The weakness of user-oriented coarse-grained communication is that we cannot use formal tools and models used in classical component architecture. The tools in fact are tuned for the use of fine-grained interfaces. It is yet another topic of further study.

References


Jarooslav Král received MS degree in statistics, CSc (about PhD) and DrSc. (one level above PhD) degrees in parsing from Charles University, Czech Republic. He is currently Professor at Charles University Prague and Masaryk University Brno. He is the author and coauthor of three books dealing with problems of software engineering. He has written more than one hundred and fifty papers on formal languages, compiler construction and parsing, random number generators and simulations, education, and on (current main interest) service-oriented and soft real-time systems. He has taken part in several projects including compilers, manufacturing systems, and additive random number generator.
Michal Žemlička received BS degree in computer science in 1992, MS and PhD degrees in software engineering in 1994 and 2006 – all from Charles University, Prague. Currently he works at Charles University as senior assistant professor. His current research interests include information systems development, software engineering, data storing, indexing, and compression, software developers education, secure gathering of publishable information from sensitive data, train communication and complex path searching.

Copyrights © 2011 KAIS