Service Orientation in Environmental Information Systems

Jaroslav Král¹ and Michal Žemlička¹

Abstract

Environmental information needs and environmental processes must reflect the fact that many actors of very different types and interests are involved in environmental activities. The actors can be international bodies, government offices, municipal institutions, (global) enterprises, citizens (often without any deeper information technology knowledge) and their activity societies like Greenpeace. Under these conditions any environmental information system (EnIS) must be able to support collaboration of existing information systems and other software systems. It must be opened for new actors and able to use new software systems. It should further support environmental processes (e.g. actions during environment accidents). We show that all these requirements can be realized if EnIS are service oriented and have specific service-oriented architecture (SOA). EnIS therefore can use experiences, skills, and tools related to service oriented systems. Examples are e-government, information systems of global enterprises. It is even possible to apply commercial tools like Enterprise Service Bus or Enterprise Service Architecture. We will discuss the crucial points of SOA and different variants of SOA applicable in EnIS. We show how the quality of environmental data can influence the architecture of the system.

1. Introduction

Environmental research, environmental activities (environmental data collection and presentation, greenhouse problems analysis and assessment, various emergency actions due the various environmental accidents, legislative environment related processes, activities and agreements of international bodies, etc.) involve a broad group of actors of very different types. The actors can be international bodies, government offices, municipal institutions, (global) enterprises, citizens (often without any deeper information technology knowledge) and their activity societies like Greenpeace. Many software entities, e.g. IS of institutions must behave like actors. The actors must (should) collaborate and provide their data to all interested (sometimes also authorized) parties. The actors can require responses from (or services of) other actors, can take part in various (business) processes based on workflow management tools or specifications based on networks of activities. They therefore often have some features of soft real-time control systems. It has very important consequences for data quality. Almost any environmental information system (EnIS) must be able to support collaboration of existing information systems and other software systems.

Such systems must be opened for public and new actors. It should further support environmental processes (e.g. networks of emergency actions during environment accidents or the allowance procedure needed for the permission of highway construction).

The actors can be subjects of very different properties. They can be based on machine bureaucracy, professional bureaucracy, ad hoc bureaucracy or they have no explicit organizational structure. The actors are as a rule autonomous. Many actors and other interested subjects have their own information measurement

¹ Charles University, Faculty of Mathematics and Physics, Department of Software Engineering. Malostranské nám. 25, 118 00 Praha 1, Czech Republic. e-mail: jaroslav.kral@mff.cuni.cz, michal.zemlicka@mff.cuni.cz. This research was partially supported by the Program "Information Society" under project IET100300517 and by the Grant Agency of the Czech Republic under project 201/03/0911.
or control systems. They use specific data structures able to support various functions, not only the environmental ones. The systems should be permanently active a must be able to work independently.

Another problem is that the computer literacy of the end users of environmental information systems is very different and often quite low. Last but not least the quality of environmental data is not excellent.

We show that all these requirements can be realized if EnIS are service oriented and have service-oriented architecture (SOA). A system has SOA if it has structure formed by a virtual network of asynchronously communicating software components and encapsulated real world bodies called (software) services. The services behave like the real world services in mass service systems. We will discuss the crucial points of SOA and different variants of SOA applicable in EnIS. SOA is the only known way of effective implementation of the above requirements. The implementation can be effective and user friendly especially in the case we take the advantage from the fact that EnIS can benefit from the fact, that the actors or software components know each other.

The quality of environmental data can influence the architecture of the system. SOA simplifies the implementation of user oriented interfaces of services.

2. Environmental information systems and service-oriented architecture

Environmental information systems must be able to interconnect (integrate) almost independent or autonomous large software components (legacy systems like the information systems of various enterprises, third party products) working as permanently available (on-line processes) activities having a lot of common with real world mass services. Such software components are called (software) services.

The corresponding software systems must usually have a structure peer-to-peer network of large components (i.e. the system has a coarse granularity and from the point of view of system architecture all components are equal) communicating asynchronously via messages of appropriate structure or via data-stores.

The messages are transported by a middleware (CACM 1995, Král/Žemlička 2002b).

![Diagram of SOSS Architecture]

Fig. 1: The structure of an SOSS
A closer look indicates that an important feature of SWC is the stability of the communications between the communication parties. Let us give some examples:

1. Information system of an international enterprise having the structure of a network of autonomous organizational units (divisions). The information system of the enterprise is formed by a peer-to-peer network of information systems of the divisions and by some additional units serving the whole system of the enterprise. The messages have now usually the format defined by an XML dialect. Note that such architecture simplifies the integration of new divisions and purchased enterprises as well as the selling out some divisions, insourcing and outsourcing of some activities.

2. Information system of e-government must be built as a network of the information systems of particular offices.

3. Environmental information system connecting people, enterprises, offices, institutions or international bodies. The kernel of EnIS is formed by permanently collaborating parties and their software components in the way similar to e-government systems (Král/Žemlička 2001).

4. The collaboration between the information system(s) of the given enterprise and the information systems of suppliers as an implementation of supply chain management (SCM, Lowson/King/Hunter 1999) and customer relationship management (CRM, Dyché 2002).

5. The information system supporting an open purchase coalition of car vendors. The aim of the coalition is to better the position of the coalition members with respect to the suppliers of the automobile parts and to increase the size of orders.

6. A highly open association of health organizations (physicians, hospitals, laboratories, database services, etc.) forming an information system intended to simplify, enhance and speed up health services.

7. Open dynamically changing network of cooperating software systems necessary for the running of e-commerce.

In all the above cases the system is a peer-to-peer network of a highly autonomous components used as black boxes (i.e. their interface only is known).

The cases 1 and 2 describe the situation when the network of services is comparatively small and not too dynamic. The communicating parties are tightly related. It results into the situation when a quite complex user oriented message (proprietary) formats can be required and successfully used. The formats of messages tend to be suited to knowledge domains of the enterprise staff and to the requirements of the enterprise activities. It often leads to complex and 'declarative' proprietary usually XML related data formats. We shall call such systems *confederations*.

In the cases 4 and 6 any member of the network often must look for a partner (addressee of a message) at the start of the communication but the collaboration is rather long term.

In the case of e-commerce the partners must be looked almost always for. The search may be performed all over the world. It implies strong emphasis on universal world wide standards. It follows that such systems must obey international standards like SOAP/WSDL (W3 Consortium 2003, 2001a) and use global network, now Internet. We call such systems alliances.

The SOAP protocol must be universal. As such it must be based on low level (i.e. programmer oriented) and not declarative and user oriented programming concepts. It forms a barrier for the involvement of end user in the specification and modification of alliances. We shall show that it is an unpleasant drawback for the definition, modification and control of business processes. User orientation is not of interfaces is not critical, it is difficult to be achieved.

---

2 Other solutions are not feasible (Král/Žemlička 2001, 2002b, Rowe 2002).
EnIS has some functions of e-commerce, but the central activities are supported by a network of services having the properties of confederations.

3. Software confederations in details

Software confederations are appropriate and often must be used e.g. in following cases:

1. Information systems formed mainly as a coalition of information systems of organizational autonomous subunits. Examples are:
   - information systems of e-government (subunits – offices – having their information systems; the offices want to use their systems without substantial changes; they also want to “own” their systems in the sense of software engineering);
   - Health care information systems (subunits are almost independent health care units like hospitals, physicians, laboratories, etc.);
   - Information system of a decentralized enterprise (subunits are autonomous divisions);
   - The kernel part of EnIS (IS of institutions and enterprises responsible for environmental politics and environmental activities).

2. Software coalitions developed via decomposition of existing systems. Typical cases:
   - Reengineering of “worn out” systems tending to be too instable after a long lasting maintenance.
   - The functionality must be enhanced and new requirements imply the change of the system architecture; this seems to be the case of the new systems by SAP or Oracle.

3. Software developed as a coalition from scratch (some real-time systems or large systems being so complex it is not feasible to develop them as system having a SOA rather monolithic architecture).

Software confederations usually include newly developed (infrastructural) services providing:
   - Flexible user interfaces (portals) of the whole system;
   - Flexible user oriented service interfaces provided by specific (usually newly developed) services (peers) like the front end gates described below;
   - Services enhancing the power of the used middleware and the stability of service interfaces.

Software confederations were in a less sophisticated form (probably) firstly used in soft real-time systems especially in flexible manufacturing systems. Manufacturing is the domain of human activities with (due the information technology) the largest growth rate of the labor productivity (Landauer 1995). It was a consequence of the fact that on the operational level the effects of the application of information technology are well measurable and quite well understood. The second reason was that the information systems supporting the manufacturing (see e.g. CIM, Scheer 1994) had to have a specific architecture, the architecture of software confederation, having important synergic effects.

This fact is not often recognized as important (Scheer 1994) and its consequences and promises for the integration of the manufacturing information system into systems of the higher levels of management/control are not used enough. It is especially apparent in the case of the implementation of the functions CRM (Customer Relationship Management – Dyché 2002), SCM (Supply Chain Management – Lowson/King/Hunter 1999) or in the case when artificial intelligence systems are to be integrated into business information system (Aubrecht 2002).

Application services (AS) are autonomous (and as a rule autonomously developed) permanently working processes/applications (permanent services) that are used as black boxes (their internal structure is not known or the information is not used). Application services can be either newly created or they can be existing (legacy) systems or third party products.
For the contact with its environment any AS must be equipped by a gate providing access to all the functions that can be available to other parties of the confederation. This entry point of a component is called primary gate (PG). PG should potentially offer all the functions and/or data of the corresponding application service. Under this circumstances PG developers tend to design the interface (the formats of messages) in such a way, that it disclose at least the implementation philosophy of the AS (e.g. that it is written in object-oriented style). It has some substantial drawbacks (see e.g. Král/Žemlička 2002a for details).

User gates (UG) are user entry points of the confederation. It is advantageous to create UG as a service. We call then UG user interface service (UIS). The internal structure of UIS is in contrast to AS known to the developers of the confederation (and the confederation's developers usually implement it) – UIS therefore behaves like a white box. The requirements on the interfaces of confederations change usually very quickly. It is therefore important that UIS should be easily modifiable (programmable).

The purpose of UIS can be defined as a facility transforming user messages into the messages directed to various application services the confederation. UIS collects answers from application autonomous components into integrated answers to the user.

If a communication inside a confederation is based on the message formats defined via XML dialects then the task of UIS is a transformation between HTML and XML message formats or between different XML dialects. It is possible to use XSLT (W3 Consortium 1999b) for this purpose.

Suppose that we want to have access to all functionality of an application service. Then we should be able to use all its low level commands or methods. Such commands are usually implementation-dependent: If the component is written as object-oriented (OO) program then its interface tends to have syntax of (remote) procedure calls; if the component is a relational database then its interface tends to have syntax able to express SQL commands (W3 Consortium 2005). This is unpleasant for the partners of the components (i.e. the components communicating with it) as it is likely that they will have to react to the changes of the implementation of the services. More important is that such formats are implementation-oriented and that they do not reflect the knowledge domain of the users.

4. Software engineering advantages of service-oriented systems

Service-oriented software systems (SOSS) must have the structure of virtual peer-to-peer networks of autonomous software components (usually complete applications). SOA is especially advantageous for business information systems (compare e-commerce) and decentralized organizations. The services are permanently active processes and the systems behave like real world mass service systems. SOA provides no inherent explicit tools for the definition and control of business processes. SOA enables new ways of implementation of (environmental) business processes in SOSS effectively using the techniques enabled by service orientation. Such an implementation reduces the use of centralized services, if any. The kernel idea of the implementation is the generation and use of newly generated services having specific roles and structure. Service-oriented software systems (SOSS) and especially software confederations give great opportunities to provide functions of new types and can provide many software engineering advantages, namely:

- modifiability (changes tend to be local, new components can be easily integrated, different SOSS can be easily interconnected, etc.);
- large and complex systems built using SOA need less effort (Král/Žemlička 2003) to be developed than systems built using monolithic approach (it is one application only);
- openness (easy to connect other systems or integrate new services);
- support for selective reengineering (of some components only) and outsourcing (some services only);
• simplified restructuring of the organizational structure of enterprises (e.g. decentralization);
• simplified business process reengineering (BPR) and the applicability of agile forms of development for the construction of big systems;
• effective implementation of environmental control processes and environmental business processes.

5. Requirement Specification Issues in Software Confederations

Requirements specifications of SOSS and therefore of EnIS should include – according to the experience of the authors – the following points:

• It must be explicitly stated that SOSS is to be developed.
• It should be specified which components/services types will be integrated into the application network (the service implementation type is important – data service, activity, supporting middleware, encapsulated people (i.e. interface to certain users eventually substituting a service or providing a prototype of it), legacy system or third party product, etc.).
• Component services/functions should be specified on declarative and user-oriented level. Preferably, the specification of peers should be formulated via message formats obeying a user-oriented language that defines the interface of the components.
• Principles, tools, and services of the infrastructure (middleware) are specified.
• It is defined how the peer services are connected through middleware.

The prevailing form of communication in SOSS is command/operation. Hence it is usually not necessary and even possible to build a central data store but often there are also technical reasons why to avoid it. The structure of SOSS is in Figs. 1 and 2.

6. Front-end gates.

For the needs of the communication with other services we therefore need an interface being problem-oriented and not implementation-oriented. So there is a need of a transformation between the two message format types. To this purpose we can use a yet another type of services – front-end gates (FEG). FEG is again an autonomous application with the structure similar to user gates: From the developer's point of view it is a white box and a node of the peer-to-peer network of components forming the confederation. Front-end gates play several roles:

• They stabilize and generalize the interfaces of components being application services – the interface based on a problem-oriented language (e.g. in an XML dialect near to the language of the knowledge domain of the partners) is very stable. It is then possible to change an AS $C$ and modify its FEG so that other services of the confederation are not influenced by the changes in $C$.
• FEGs allow creation of several different interfaces of the given AS when different groups of users/partners of it need different services or different interfaces. Different FEGs can e.g. provide different security levels or different middleware services for different partners.
• FEG allows implementation and use a user oriented declarative interface of a service. It is crucial for the feasibility of the requirement that business processes are to be defined and modified by users.
• FEG can provide many other services. It can provide security functions or it can help to control business processes (see below). It can help to change the communication type from the communication via a data store into purely message driven (on line) one and vice versa.
Front-end gates can use the same technologies as user interface services in simpler cases XSLT, in the more complicated ones (e.g. if there is a need to optimize output messages), the techniques known from the construction of compiler back-ends (Aho/Sethi/Ullman 1986).

It is important to point out, that the services in confederation need no be web services and the middleware need not be Internet. It is even possible, that the middleware uses different protocols. The communication between front-end gates and application services should not be assessable to public.

7. **Environmental business processes in SOA**

The (workflow) environmental processes are in fact the business processes in common sense. We therefore will discuss the environmental (work) processes as a specific case the case of business processes. It should fulfill the following requirements:

1. Every process should have its owner responsible for the business consequences related to process enactment. It is necessary to supervise the process progress and to require responsibly of various actors (punishing people responsible for a wrong decision).
2. The owner can instantiate the process and often he/she can define it. In any case the owner should be able modify the process on the fly to respond to the new facts occurring during the process execution (emergency responses).
3. Various models defining the business process should/must be used as there are many business process modeling and control tools, e.g. ARIS (IDS Scheer), business rules, BPDL/BPEL languages (Andrews 2003), Petri nets model (Petri 1962), workflow specification system, activity diagram from UML and even the simple text and box driven systems. All he variants are likely to be used under different circumstances.
4. It is good to have a database of business process templates (definitions) but it should be possible to work without it.
5. The process definitions and process modifications should be made by system users without contact with systems developers.
6. It is desirable to develop and test the business processes autonomously without presence of the central database of process definitions.

![Diagram of process manager generation and use]

The process definition must specify a network of activities provided by the services. This information can be modified due to the emergency responses and sometimes the state of the process must be somewhat coded. Like in the human society we have several possibilities:

1. There are no data outside the services defining the process and its states. The process is encoded into behavior patterns of application services or into their front end pages (i.e. each service alone 'knows' how to support the process, what service provides the next activity of the process). This is preferable in alliances. The main engineering disadvantage is that the changes of business processes imply changes inside of services being black boxes.
2. The states and the structure of processes are included in messages/data stores. This model is more flexible but the changes of processes must again be implemented by services or their front-end gates (like in point 1).
3. There is service (process manager) defining the process and storing its state(s). It is desirable that the service is not central. The best solution seems to be to generate a new process control service called process manager (PM) for every new business process. The control service can be on-line modified by human management when necessary.

The life cycle of a business process then consists of three steps (see Fig. 3):

**Process instantiation (enactment).** A new service $P$ of the type process manager is generated. It uses process model (definition). $M$ is usually read from a database of process definitions. The models can be written in process definition language BPDL. The definition of the process can be defined on-line by the process owner, if necessary. $M$ is transformed into process control data structure $C$ by adding some parameters (time, resources, etc.). $C$ can be specified in BPEL language.

**Process execution.** Application services are called asynchronously via sending messages or via data stores. The process execution is supervised by process owner. The process control data structure $C$ is modified e.g. by process owner if necessary.
Process retirement. The process is finished. It can generate the information about the results of the process. It enables the calls of business processes inside business processes.

It is not difficult to see that the implementation can fulfill the above specified requirements on business processes. Note that the manager can have the same communication protocols for different process control data.

The implementation of business processes is easier in confederations than in alliances as we can use application specific communication protocols and there are no problems were newly generated services can reside. It does not imply that the above solutions are not applicable in alliances. It is simply more difficult and there is less experience. Note that the process manager hides the process control data and enables a great freedom in of process modeling frameworks.

8. Seeming Simplicity
One of the most surprising facts related to service orientation is that the principles of it are routinely used in operation systems (I/O drivers) and real time systems since the invention of minicomputers. The principles of service orientation are quite simple. There is therefore no need to develop any sophisticated theory of service orientation – at least for time being. In contrast to its seeming simplicity the service orientation is very likely to be able to open the way to software as a high-tech product in common sense, it is, a product with a satisfactory reliability assembled from the parts from various sources according to standardized and reliable assembly procedures. What are the reasons for this disagreement?

1. Up to now the main problem was to develop reliable applications. The communication between two applications was not any crucial issue.
2. There were no satisfactory tools and standards like Internet to support the communication.
3. The switch to service orientation is not easy as it enables actions; service-oriented systems are not information providers only. The main attitudes of service orientation are different from attitudes and philosophy of revealing object orientation. There are cases that antipatterns in object-oriented philosophy become very good patterns in service orientation (an example is Island of Automation). Service orientation is under these conditions an unpleasant novelty for many software professionals.
4. Service orientation at least for time being is not a stimulus of sophisticated theories, so it is not interesting for academia.
5. There are prejudices against tool and skills change. It is supported by the current computer curricula.
6. Service orientation requires a close collaboration with users. It is not favorable for many IT graduates.

All these issues are in fact a consequence of the fact that service orientation is a specific paradigm. It is not known for many users and developers. And they are not used to. The change to service orientation is then a paradigm shift. It is always a painful and long-term process.

9. Data Quality
Environmental data are generated by various systems in various formats. Their relevance and meaning can be (and is) doubted. In other words the quality of environmental data is not the best one.

It is known that data quality like timeliness can substantially change the principles and the structure of software systems using them. For example, the fact that the durations of steps of a project activity network
are not accurate enough implies that popular critical path method (and applications using it) cannot be used (Goldratt 1997).

Quality is, according to ISO 8402:

Characteristics of an entity as a whole that give the capability to satisfy explicit and implicit needs:
- Quality of an entity is a subjective concept dependent on requirements that the user of the entity requests in an implicit or explicit manner.
- Quality is a multidimensional concept tied to various characteristics.

The problem with data quality is that there is no generally accepted concept of data quality and that there is no generally accepted concept of information quality different from the concept of data quality.
As mentioned above, the data quality is a multidimensional concept.
The metrics of data quality are usually fuzzy (given by finite ordered sets, for example: poor, not too good, quite good, good, almost excellent, excellent) or numerical (e.g. weight, all arithmetic operations are allowed).
The most used data quality characteristics are (according Strong/Lee/Wang 1997):

1. Intristic metrics – properties of data themselves.
   a. Accuracy.
   b. Objectivity – measure of data bias.
   c. Believability.
   d. Reputation.
2. Accessibility and access security.
   a. Relevancy.
   b. Value added.
   c. Timeliness (time for data update).
   d. Completeness (all data or data subset with unbiased metrics exact enough).
   e. Amount of data (enough for given purpose).
4. Representational. Ease of understanding, combining and presentation.
   a. Easy of understanding.
   b. Interoperability.
   c. Concise representation.
   d. Consistent representation.

It is clear that many data quality dimensions must be based on subjective judgments (must be subjective, the metric that can be measured in classical sense are objective). To make the judgment less random and less subjective a specific process must be introduced (see www.DataQualityAct.us or NRS State Data Quality Standards Checklist http://www.doe.mass.edu/acls/mailings/2002/1213/NRS_check.pdf).
The problems with data quality can reduce the applicability of Semantic Web (W3 Consortium 2001b) (and other popular concepts) – especially in the case when the quality of data from various sources varies from source to source. Central tool of semantic web is the language RDF (W3 Consortium 1999a). RDF does not have enough powerful tools to deal with data quality.
If the data are not accurate, some data are missing and are not timely, we cannot request services online, it is without involvement of managers and their intelligence. In this case the parameters of service requests must be stored and be accessible for editing by managers. It is quite common in e.g. small machine building enterprises. Data stores are almost unavoidable when there are multiple destinations (e.g. several equivalent services)
It is not generally understood what is the difference between the data quality and information quality. The trend is to consider data quality and information quality as different (but related) concepts. The
information quality is now often understood as an entity with lifecycle starting from vision through requirement specification, implementation, use, and retirement (IQ Conference www.iqconference.org).

Current leading institution in data quality and information quality research is MIT (http://mitiq.mit.edu/).

10. Conclusions.

SOSSs offer a feasible solution of many software engineering and software management problems. It simplifies outsourcing, software reuse and enterprise decentralization. It is the only known feasible way how to build environmental information and control systems.

It enables a very powerful solution of environmental business process definition, enactment, and control. There are yet barriers to be overcome. The proper use of service-oriented systems requires a new way of thinking of software developers, software vendors, and last but not least the end users themselves. It requires new attitudes of the management of software vendors as well as of the users. All of them must accept the principle, that things cannot be ready at once and that old fashioned applications can be successfully reused.

The principles of service-orientation are very successfully used at commercial level. Examples are Enterprise Service Bus (ESB) by Sonic Software and Enterprise Service Architecture (ESA) supported by SAP and by other dominant software producers. It is important to accept the fact, that EnIS have some properties of control (real time) systems and that basic requirements on the behavior of EnIS are typical for many existing systems. So we can and should use many existing tools developed for enterprises (like ESB or ESA).

Service orientation is however not fully accepted by academia. There are several further issues. The most important is the fact, that we have no good modeling tool for SOA systems yet.

Bibliography


