FLEXIBLE SERVICE BINDING IN DISTRIBUTED AUTOMATION AND CONTROL SYSTEM

Cristiana Voican

University Politechnic of Bucharest,

Email address: voicancristiana@yahoo.com

Abstract. Particular emphasis was placed on the dynamic lease-based binding of services which on the one hand provides flexible and loose coupling of system components but on the other hand has to ensure reliable communication and cooperation. The guidelines were applied to the experimental implementation of a manufacturing cell control system using a real-time version of the Java Runtime Environment.

The Device Profile for Web Services (DPWS) was used as basic infrastructure technology. Test and evaluation were performed under distributed simulation of technical processes and devices

1. INTRODUCTION

One of the key features of service-orientation is the use of loosely coupled components. As all devices, sensors and actuators provide a service interface the coupling of components can correspond to the flexible binding of services.

This flexible binding of services demands for service description, discovery and selection, and service association and linking mechanisms.

The service description subsumes three basic parts:

• Type and interface definition,
• Binding and communication information,
• Functional properties.

The type and interface definition of a service specifies the methods and parameters associated with a specific service type. All services that comply with a specific service type offer the same interface.

The binding and communication information contains information about the actual communication endpoints and the basic communication mechanisms, such as IP addresses and ports, and application protocol regulations. At last, the functional properties complete the information on devices in the automation system. They e.g. include, which sensor is attached to which conveyer and what is the exact position.

The service description is the basis for the discovery and selection of matching services by the automation process and control services. In our system, the discovery and description phase are based on DPWS technology and thus adhere to the WS-Discovery and WS-Transfer (for metadata exchange) standards.

2.. FLEXIBLE SERVICE BINDING

The association and linking of matching services with a particular client is handled by our lease based binding approach to meet the requirements of a flexible but also stable way for dealing with loosely coupled services in the domain of industrial automation.

The notion of a lease was first introduced by and was used to provide an efficient, fault tolerant way for using file caches in distributed environments. Further on leases were used in Jini to grant clients access to network services.

In the case a client wants to use a particular service, it issues a lease-request which contains a duration for which the client wants the lease to be valid.

The service responds with a denial or a grant. A granted lease is valid only for the duration.

Thus the client has to request another lease for service use after the current lease has expired or may prolong it before its valid duration has passed.

In automation systems a client usually uses a set of services (sensors, actuators, and controllers) and has to allocate a suitable ensemble.

Therefore we extend the lease model by adding support for the atomic allocation of service ensembles.

The atomicity property guarantees that a client either is granted the leases for all requested services or it gets no lease at all. This atomicity is achieved by a 2-phase algorithm, which is similar to the 2-phase-commit protocol. It is a lease granting algorithm with explicit reservations (cf. Figure 1).

During the coupling phase the client asks the suitable services for reservations. Reservations are binding for a short duration. If all services positively respond, the client submits lease-requests that yield to valid usage leases.

If at least one service cannot satisfy the reservation request, the client cancels all other reservations.

After the coupling phase is completed, the interaction of coupled components starts. The client process configures and initializes the services and finally starts production (cf. Figure 2).

When the leases are about to expire, the client either issues a prolongation request to extend the production phase or stops the services and performs cleanup operations.

The prolongation of existing leases uses the same 2-phase algorithm as used at initial lease creation. In the decoupling phase the expired leases are fairly released and deleted.

3. APPLICATION EXAMPLE

The service-oriented control software presented so far was experientially evaluated for an example industrial automation setup. The example system and the tested applications scenarios are presented in this section.

3.1. EXAMPLE STRUCTURE

The structure of our evaluation example is depicted in Figure 6. The work pieces enter the system through conveyer conv1 and conv2.

Both conveyers are located next to a rotary disk, which is able to collect work pieces from either conv1 or conv2 by rotating the disk and using the conveyer element conv3 on top of the disk.

This conveyer transports the work pieces to conveyer conv4 which in turn moves them through the lacquer machine.

After being painted by the lacquer machine, the work pieces are checked by a laser sensor.

Inaccurate pieces are pushed into a disposal box by a pusher. Proper items are moved out of the system to the next work station. The devices and sensors (not depicted) are exporting services as described in section.

The logical control of the conveyers is implemented using a PID controlling algorithm which could be differently parameterized for evaluation purposes.

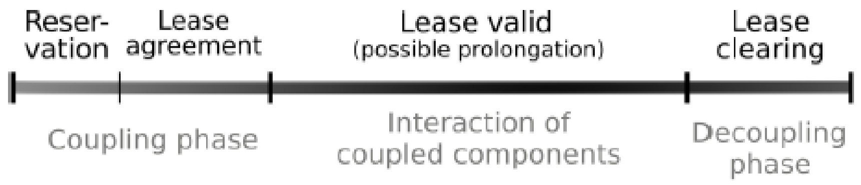


Figure 1. Lease lifecycle

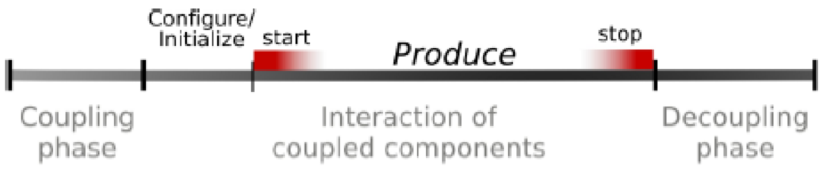


Figure 2. Leases and production

3.2. APPLICATION SCENARIOS

The example system was evaluated using different application scenarios. The scenarios use different service hierarchies and thus model different levels of control in the application process.

The first scenario comprises the following process:

1. Work pieces are picked up from conv1 or conv2.

2. The rotary disk and conv3 transport the work pieces to conv4.
3. The lacquer machine paints the work pieces.
4. The inaccurate work pieces are detected and pushed into the disposal box.
5. The acceptable work pieces are moved out of the system.

The service hierarchy for this application process is depicted in Figure 4. The application process uses six different control services (light gray), each responsible for a specific part of the example system. The control services themselves are using a set of sensor and actuator service interfaces to interact with the hardware at technical process level (dark grey). In contrast, the rotdisk control service for controlling the rotary disk and conv3 on top of the disk as a whole uses the control services of the single components. It implements an algorithm for the balanced use of the two attached input conveyers.

The second application scenario uses only the input conveyer conv1, thus the usage of the rotdisk control service is not necessary:

0. Statically move the rotary disk in conv1-conv4 position using the disk service.
1. Conv1 transports the work pieces to conv3.
2. Conv3 forwards the work pieces to conv4.
3. Conveyer conv4 moves the pieces through the lacquer machine.
4. The pusher sorts out erroneous pieces.
5. Acceptable items leave the system.

The service hierarchy used for the second scenario is depicted in Figure 5.

The application process of the second scenario uses seven control services. The sub-component services of the rotary disk now are directly used to initially set up the right direction of the disk and to control the conv3 at runtime. This change in the process outline does not infer changes in the service implementations of the devices used.

Further scenarios were used to evaluate the applicability of multiple application processes, each controlling a part of the overall process.

4. EVALUATION

The evaluation environment comprises three major components: the DPWS stack, the Java Real-time VM and the simulation system.

The WS4D.org DPWS stack, developed by Dortmund University and Materna, is a Java based implementation of the DPWS protocol stack and provides a service oriented communication infrastructure.

It was developed with modularity and extensibility in mind and thus can be adapted to varying application scenarios, ranging from small client-only implementations for mobile phones to multimedia or file-sharing services for embedded settop boxes.

The Java Real-time System comprises technologies and concepts for correct reasoning about the timing of Java real-time applications. It contains new types of real-time threads, memory handling schemes preventing the garbage collector from influencing the runtime behavior in a nondeterministic way), high precision timers with nanosecond resolution and direct memory access for implementing device drivers purely in Java. Nevertheless, the Java RTS depends on the real-time capabilities of the underlying operating system.

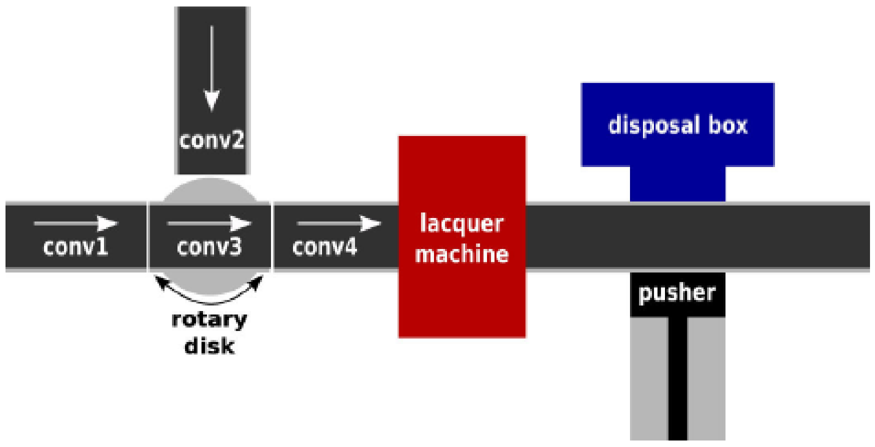


Figure 3. Example system

For evaluation purposes we developed a testing environment, split into two blocks: a simulation system and the sensor, actuator and control service implementations.

The time discrete simulation system is composed of four major components.

The simulation model component manages a grid model for locating devices, sensors and work pieces in the system and a component model for preserving the state of the simulated components.

The simulation control component periodically updates the model information.

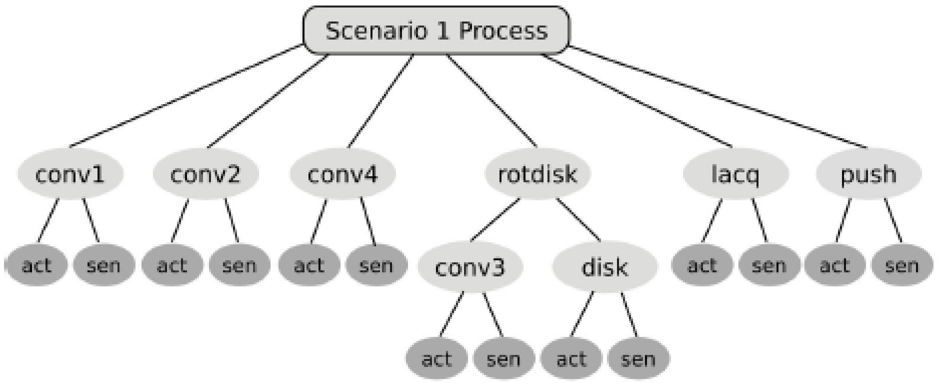


Figure 4. Scenario 1 service hierarchy

Changes in the internal state of sensors and actuators are sent to and received from the distributed components via an UDP based communication protocol. It was especially designed to consume few network bandwidth.

A graphical user interface is used to track and control the simulation.

The simulated system comprises sensor, actuator and control service implementations. The sensor and actuator implementations are connected to the simulation system via the UDP based communication protocol (s.a.) to receive and publish state information.

The simulations were run on an Athlon64 X2-3800 machine with two GB of memory and an OpenSolaris installation as basis for the Java RTS.

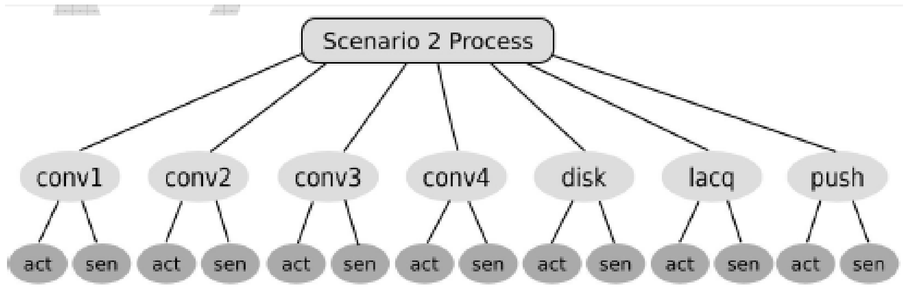


Figure 5. Scenario 2 service hierarchy

4.1. EXPERIMENTAL RESULTS

A series of experiments focused on the evaluation of the functional behavior of the control system. Particular test sequences checked the feasibility and stability of the lease-based allocation. Atomic allocation and setup of service ensembles were as well tested as atomic lease prolongation and occasional aborts followed by the searching and switching to alternative ensembles.

In the course of additional experiments the service call roundtrip times (using simple input and output parameters) were measured in order to check the current real-time limits of Java VM and DPWS based control system implementations.

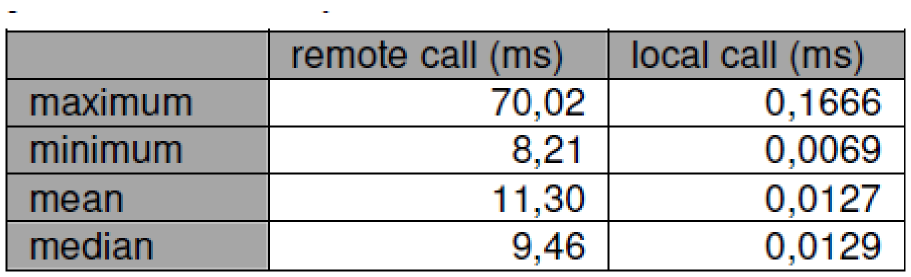
Table 1 presents the values obtained for local VM-internal (on the OpenSolaris host) and for remote DPWS-based service calls (between the OpenSolaris and the PC host).

The configuration was able to support low to medium realtime requirements (e.g. cycle times >50ms).

5. CONCLUSIONS

We have presented a service-oriented control architecture for automation systems. The architecture forms a service hierarchy ranging from low-level sensor and actuator services, over a number of control service levels up to application processes. Instead of statically associating services for the different client operations, a flexible lease based binding approach is used.

Table 1. Action call roundtrip times



	remote call (ms)	local call (ms)
maximum	70,02	0,1666
minimum	8,21	0,0069
mean	11,30	0,0127
median	9,46	0,0129

This approach follows the loosely coupled nature of components in service-oriented architectures. The algorithm used for the flexible binding approach was tested in different application scenarios.

The evaluation results regarding action call roundtrip time exhibit that the Java-based service-oriented approach may not yet be a feasible solution for all applications. However, the applicability can be extended by using e.g. hardwarebased message processing and real-time capable network infrastructures .

6. REFERENCES

[1] H. Smit, F. Jammes, “Service-Oriented Paradigms in Industrial Automation”, IEEE Transactions on Industrial Informatics, Vol. 1, No. 1, pp. 62-70, 2005.

[2] C. Gray, D. Cheriton, “Leases: an efficient fault-tolerant mechanism for distributed file cache consistency”, ACM SIGOPS Operating Systems Review, Vol. 23, Issue 5, pp. 202-210, Dec. 1989.

[3] Universal Plug and Play (UPnP), http://www.upnp.org, 1999.

[4] Devices Profile for Web Services (DPWS), http://schemas.xmlsoap.org/ws/2006/02/devprof/, 2006.

[5] Service Infrastructure for Real-time Embedded Networked Applications (SIRENA), http://www.sirena-itea.org, 2006.

[6] Sun Microsystems, Jini, Network Technology, http://www.sun.com/software/jini, 1999.

[7] Kapsers, Küfner, “Messen – Steuern – Regeln: Elemente der Automatisierungstechnik”, Vieweg Verlag, 6th Edition, p. 253, 2006.
[5] Service Infrastructure for Real-time Embedded Networked Applications (SIRENA), http://www.sirena-itea.org, 2006.

[8] WS4D.org Java Multi Edition DPWS Stack, http://www.ws4d.org, 2007.
[9] Sun Java Real-time System 2.0 (Java RTS), http://java.sun.com/javase/technologies/realtime, 2007.
[10] PROFINET, http://www.profibus.com/pn/, 2007.