

Transactum

Business Process Manager with High-Performance Elastic Scaling

November 2011
Ivan Klianov



TRANSACTUM

Transactum BPM serves three primary objectives:

- To make it possible for developers unfamiliar with distributed and scalable systems to build large-scale applications.
- To ensure that these applications handle up to 100 times growth of workload without necessity of additional hardware or software redesign.
- To maximize the availability and partition tolerance of distributed scalable application systems with guaranteed consistency and integrity of data.

Business Process Development

Transactum business process application is a set of process steps reliably connected with asynchronous messages. It exists in form of Windows service and executes on top of Transactum Process Engine, which is Windows service as well.

Transactum provides a tool for declarative programming of business process applications. It generates application's C++ source code from XML specification of its workflow graph and its business activities in form of COM components.

Thus, the only real code writing, necessary for development of a large-scale distributed application, is for programming of extremely simple COM components that provide the actual business functionality and for SQL stored procedures invoked by these components.

Transactum can build Task-oriented and Goal-oriented processes. Task-oriented processes involve execution of a set of process steps, where each step is executed only once and where the final output is predefined by the initial input and the process state.

In contrast, Goal-oriented processes might receive additional input from one or multiple asynchronously connected services and programs interacting with humans. Such processes might therefore perform unknown number of loops until the goal is achieved.

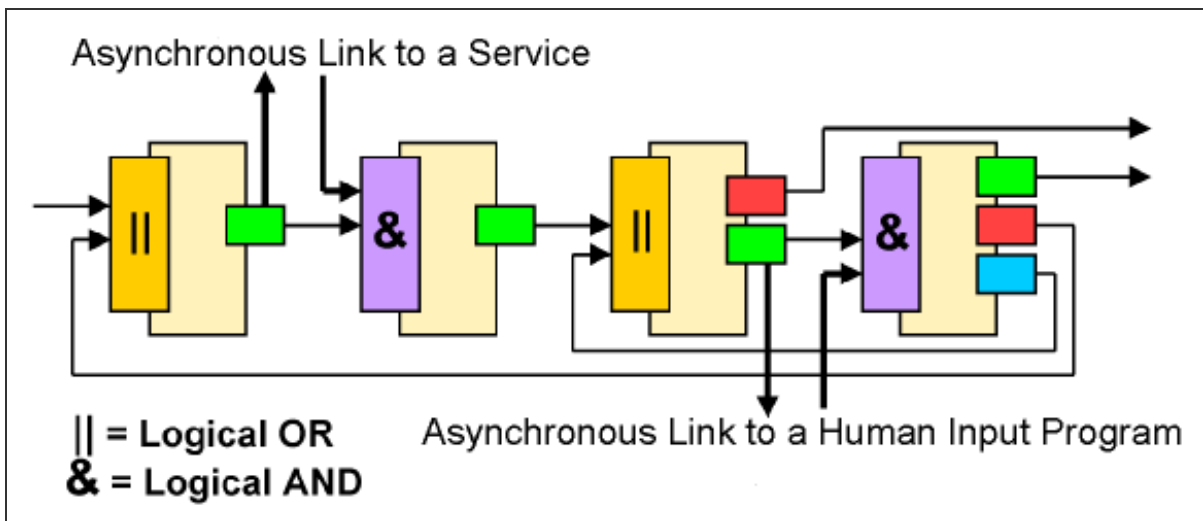


Figure 1. Transactum business process with Goal-oriented workflow

Transactum business process might split itself into multiple execution streams of parallel execution and later join these streams at one or multiple synchronization points.

The process starts with one input state, but might have multiple consistent completion states, representing the entire variety of individual semantic results of execution.

Transactum process can be a building block in construction of stateful and stateless services. A stateless service implements a client-service type of interaction with no need for multiple interaction loops with the user and no need to remember the state of the process execution with the previous loop.

Inclusion of a Link to Process User will turn the process into a building block of stateful service. The interaction between a user and a stateful service is a peer-to-peer type. It involves multiple loops of interaction and thus creates the necessity to remember the state of process execution with the previous loop.

The ability to asynchronously integrate other processes and services makes Transactum BPM the ideal tool for expansion by development of composite processes. This lets the development teams use the individual strengths of the entire variety of available skills in the context of available variety of development and execution environments.

For example, Java or C# professionals might create individual services and GUI processes and thus take advantage of Java libraries and .Net framework to ensure fast development.

These services and processes can be easily integrated into a larger Transactum business process and thus take advantage of the ability to create large-scale application with no experience in that area and let Transactum Process Engine care for scalability, consistency, availability, and partition tolerance.

Business Process Execution

Before the Internet-demanded level of scaling, the entire business logic frequently resided on the database server in form of stored procedures. This was the easiest way to deal with application consistency. The Three-tier architecture with Transactum (Figure 2) enables the business logic to be completely taken out of the database server for performance purposes, as Transactum preserves consistency with semantic invariants.

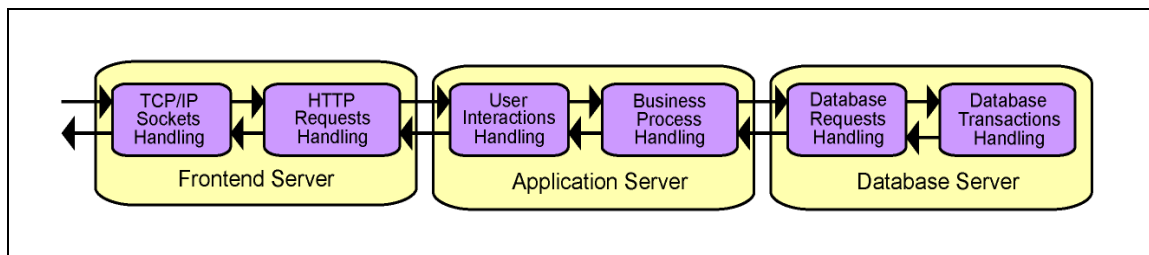


Figure 2. Transactum Web service with Three-tier architecture

Scaling a business process requires increasing the volume of its running process instances (Figure 3). A business process instance with Transactum means one or multiple threads executing one or multiple business objects in the context of a particular user request.

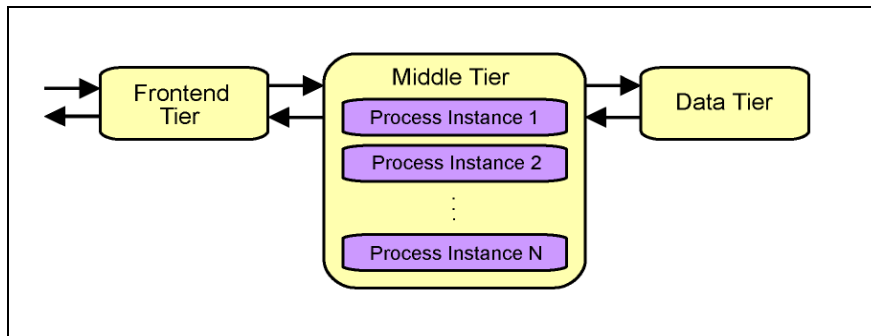


Figure 3. Scaling a business process application

Business process instances of a Transactum application run in Transactum Process Engine. An instance of Transactum Process engine runs on 2 CPU cores and executes up to 8 applications and up to 1,000 concurrent process instances of these applications. The Engine dynamically scales up (on the same hardware) and down the volume of process instance of each application according to the fluctuations of the number of concurrent users.

Figure 4 shows scaling of business process instances with Transactum Process Engine. On this figure, the number of process instances N might be as large as 1,000.

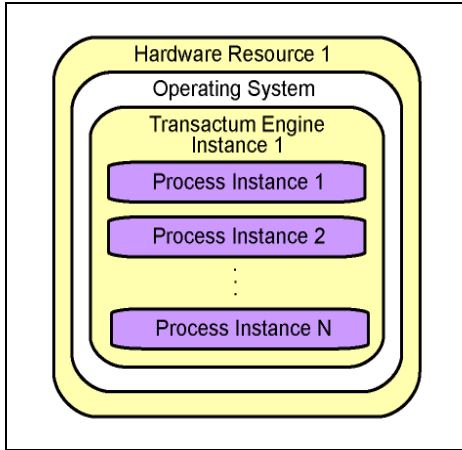


Figure 4. Scaling with Transactum Process Engine

Figure 5 presents the implementation of scaling. During the initialization of a process, a dedicated pool of threads is created per process step. Each thread instantiates a COM object with the business functionality of the step. A thread is normally in a *waiting* state before it receives an assignment to execute the process step and after the assigned work is completed.

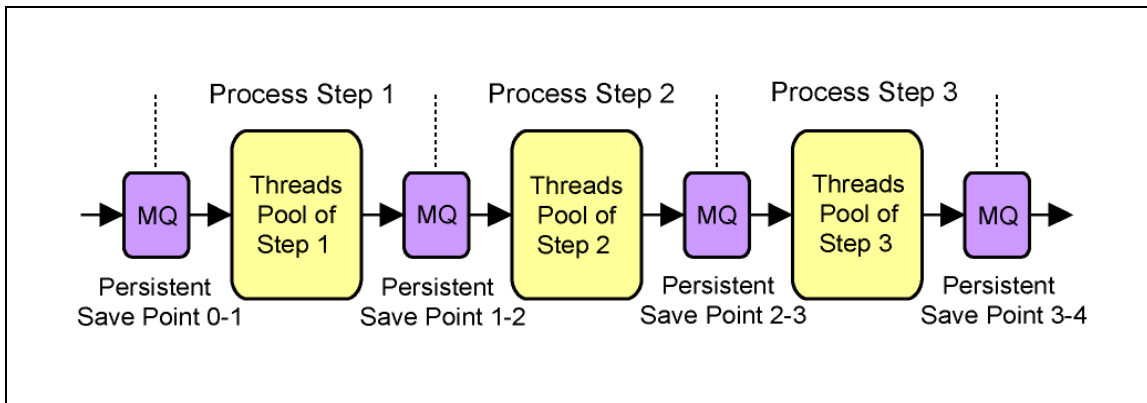


Figure 5. Transactum implementation of scaling of business process instances

A business process step starts and ends with a Persistent Save Point implemented with a transactional Message Queue. A Persistent Save Point separates every two adjacent steps of a process. Physical arrival of a message in a Persistent Save Point’s Message Queue triggers the following sequence of actions:

1. A thread from the pool is assigned to deal with the message.
2. The thread reads the message and passes it to the business object.
3. The business object performs its activities.
4. The business object composes a message for the next process step
5. On exit, the business object returns the new message
6. The thread physically fetches the old message from the Input Message Queue
7. The thread places the new message in the Output Message Queue.

Business objects of application processes executed on Transactum Process Engine perform blocking database IO calls. Compared to applications implemented on other middleware products with blocking IO calls, applications on Transactum reuse the instances of a business object at a significantly higher rate – similarly to architectures with non-blocking calls.

Mainstream middleware products with blocking IO calls create a worker thread per connection. Thus, for example, a process executing 10 steps with an object per step and serving 1,000 concurrent users requires creation of 1,000 worker threads with a total of 10,000 object instances. In contrast, Transactum implementation of scaling requires creation of 1,000 threads with a total of 1,000 object instances – 10 times smaller number of necessary object instances.

Transactum Process Engine provides multiple advantages as a business process execution environment. It performs elastic scaling in response to fluctuations in demand for higher throughput. It enables scaling of processes that perform transactions, which at present cannot be implemented on Cloud platforms. And most importantly, the scaling does not require additional hardware or redesign of software.

Consistency, Availability, Partition Tolerance

The consistency of a business process and the integrity of its data are guaranteed by the implementation of process steps as individual transactions, or as sub-transactions with higher-level coordination. Transactum gives a choice between COM+ transactions or Transactum-proprietary ACDC (Atomic, Consistent, Distributed, Concurrent) transactions.

Where the individual steps of a business process need to be executed as sub-transactions with higher-level coordination, the Two-Phase-Locking of COM+ enforces sequential performance and prevents scaling. Under a requirement for higher-level coordination (non-commutative sub-transactions due to CS-cycles or semantic reasons) ACDC transactions are the only alternative to the *eventual* consistency of the optimistic concurrency control.

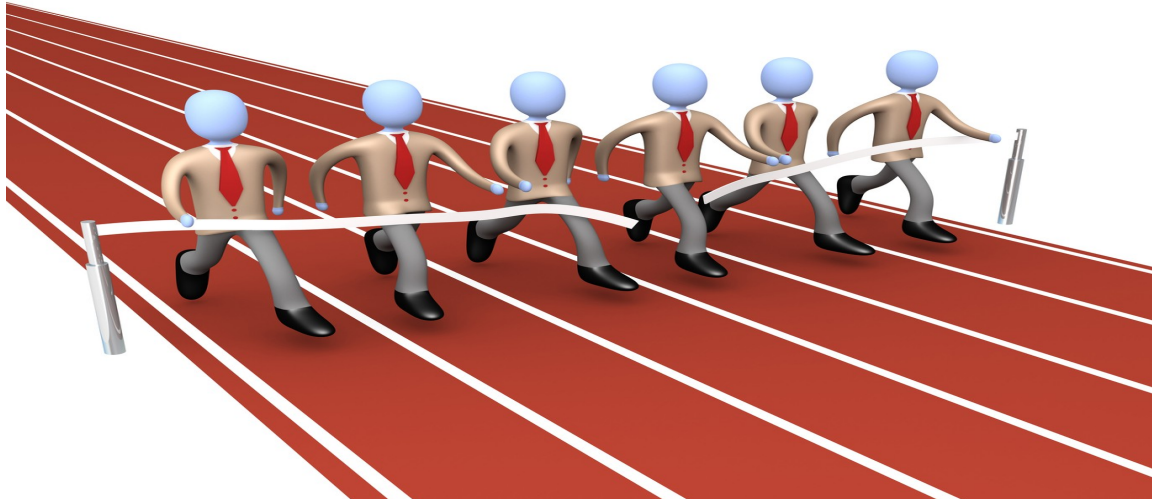
With ACDC transactions, the consistence is guaranteed with serializability schedules enforced as sub-transaction pre-execution conditions. Transactum Engine builds these invariants from XML specification of transaction syntax. Data integrity is preserved with prevented violation of constraints enforced as sub-transaction post-execution condition. Transactum Engine uses the business components to build these invariants.

The enhanced partition tolerance is achieved with Transactum software technology for real-time response to failures of network by duplication of network segments, detection of failure, and switching to the backup network segment.

The enhanced availability is based on Transactum software technology for synchronous replication of state of database nodes and asynchronous replication of state of the Process Engine. The technology is based on real-time detection of failure and switching to a healthy backup node.

Transactum Process Engine guarantees the process consistency, enhances the availability and partition tolerance of application systems, and boosts the performance per unit of hardware with two orders of magnitude, compared to mainstream BMP platforms.

Transactum Performance Summary



- 1,000 process instances per Engine instance
- Enhanced application availability with synchronous replication
- Built-In process parallelism with guaranteed consistency
- Real-time detection and response to partitioning events
- Built-In elastic scaling on constant hardware
- Pooled capacity for parallel processing
- 8 Applications per Engine instance
- Multiple Engine instances per computer, limited by available CPUs only