

# On Enhancing Component-Based Middleware with Transactions

Marek Prochazka<sup>1</sup>, Romain Rouvoy<sup>2</sup>, Thierry Coupaye<sup>3</sup>

<sup>1</sup> INRIA Rhône-Alpes, 665 avenue de l'Europe, 38334 Saint Ismier Cedex, France

<sup>2</sup> INRIA Jacquard Project, LIFL, 59655 Villeneuve d'Ascq Cedex, France

<sup>3</sup> France Télécom R&D, 28 Chemin du Vieux Chêne, BP 98, 38243 Meylan Cedex, France

**Abstract.** It is believed that transactions belong to key services of component-based middleware. However, surprisingly, there is no general agreement on how the component-based middleware support for transactions should look like. In this paper we exploit our experiences with Jironde [4], a flexible framework that extends the Fractal component model [1, 2] with transactions via a set of transactional controllers that manage transactions on behalf of a component. Then we identify several key architectural and technical issues related to enhancing component-based middleware with transactions. We investigate how current technologies address these issues, and the suitability of current standards to support transaction processing in component-based middleware.

## 1 Transactions and Components: Architectural Issues

Different component models deal with component's participation in a transaction differently. In the *explicit transaction participation*, the scenario of involving a component  $C$  to a transaction  $t$  essentially consists of the three steps as follows: 1)  $C$  is registered to  $t$ . 2) A client invokes various operations on  $C$ . 3) At the time of  $t$ 's validation, the transaction manager invokes specific methods of the registered  $C$ 's interfaces. These (callback) methods must be implemented by the transactional components. With the *implicit transaction participation*, components are not obliged to implement any functionality related to transactions. Any time  $C$  is visited by a transaction  $t$ , the transaction manager of the container keeps all necessary information to manage atomicity, concurrency control, and recovery. Different component standards deal with component participation in transactions differently. CCM use the explicit transaction participation, COM+ uses the implicit one, while EJB mix both.

*Component-unaware transactions* manipulate data without any knowledge on whether they are organized or related to components. For *component-aware transactions*, components are the data they manipulate with. We believe that all the CCM, EJB, and COM+ transactions are component-unaware. A component is *transaction-unaware* if its code does not use any transactional primitives and is not therefore in any way dependent on any transactional standard, while a *transaction-aware* component is the opposite. Component's awareness of transactions reflects the implicit/explicit transaction participation but includes also some hidden expectations of the component design. *Transaction-unaware container* does not deal with

transactions, which are managed at the application level instead. *Transaction-aware container* provides some transaction management features, such as container-demarcation, transaction propagation, concurrency control, etc. EJB, CCM and COM+ are examples that provide such containers.

## 2 Transactions and Components: Technical Issues

The technical issues related to transactional component include concurrency control, recovery, and transaction context propagation. As for concurrency control, the current technologies either use a simple read/write model or do not allow any concurrent access to component instances at all. It would be beneficial to exploit the concurrency potential of components by e.g. the use of conflict matrixes defined on all the methods of all implemented interfaces. Both concurrency control and recovery of components reflect the architectural patterns presented in the previous section. Another important issue is the transaction context propagation. There are several options of how to specify a *transaction propagation policy*, as well as whether to define it either during the component's development or during its deployment. A last issue is to support the definition of new propagation policies, like JOTDF [5] does.

## 3 Conclusion

Our experiments with Fractal and Jironde have shown that the coexistence of components and transactions raises more non-trivial architectural and technical issues than expected. The current middleware standards and technologies do not address these issues satisfactorily. So we hope that future developments will take into account such issues for transactions to remain a key service of component-based middleware. Details on our work are available at <http://jotm.objectweb.org>.

## References

1. Bruneton, E., Coupaye, T., Stefani, J.-B., "Recursive and Dynamic Software Composition with Sharing", the 7<sup>th</sup> International Workshop on Component-Oriented Programming (WCOP 2002, in conjunction with ECOOP), Malaga, Spain (2002)
2. ObjectWeb, "The Fractal Composition Framework Specification", Version 1.0, <http://fractal.objectweb.org/> (2002)
3. Prochazka, M., "Advanced Transactions in Component-Based Software Architectures", Ph.D. thesis, Charles University, University of Evry (2002)
4. Prochazka, M.: "Jironde: A Flexible Framework for to Make Components Transactional", 4<sup>th</sup> IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2003), Paris, France, to appear (2003)
5. Rouvoy, R., Merle, P., "Abstraction of Transaction Demarcation in Component-Oriented Platforms", ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil (2003)