ADAPT
IST-2001-37126

Middleware Technologies for Adaptive and Composable Distributed Components

Evaluation

Deliverable Identifier:	D19
Delivery Date:	1st October 2005
Classification:	Public Circulation
Authors:	*All partner members*
Document version:	1.0 23rd September 2005

Contract Start Date:	1st September 2002
Duration:	36 months
Project coordinator:	Universidad Politécnica de Madrid (Spain)
Partners:	Universitá di Bologna (Italy), ETH Zürich (Switzerland), McGill University (Canada), Universitá degli Studi di Trieste (Italy), University of Newcastle (UK), Arjuna Technologies Ltd (UK)

Project funded by the European Commission under the Information Society Technologies Programme of the 5th Framework (1998-2002)

Contents

1 ADAPT Evaluation

The purpose of this document is to provide an evaluation of the results of the ADAPT project, both in the area of basic services and composite services. This evaluation has been completed by following the Revised Evaluation Plan (D16), which was a month 24 deliverable of the project. The structure of this evaluation follows very closely the structure of the evaluation plan; the first section will examine the non-functional capabilities and attributes of the ADAPT system. The second section then examines the functional capabilities and attributes of the ADAPT system.

1.1 Evaluation of Non-Functional Capabilities and Attributes

The main non-function goals of the ADAPT project are in the areas of Availability, Adaptability, Scalability and Performance. This section is divided into two parts, an examination of the basic services support and an examination of composite service support.

1.1.1 Basic Service Support Experiments

The evaluation of the Availability, Adaptability, Scalability and Performance of the ADAPT Basic Service support involved analysis at each tier of its Basic Service platform. The front tier being the support for web services, the middle tier being the support for EJBs and the back tier being the support for the databases.

1.1.1.1 Via the Front Tiers

The evaluation of the front tier of the ADAPT Basic Service platform involved analyzing the performance of the JMiramare system. This was done on a cluster of four Athlon XP 2600+ boxes with 512 MB RAM each. The software present on the machines were:

- S.u.S.E. Linux 9.0;
- Java SDK 1.4.2;
- JBoss 3.2.3;
- Adapt-replication-framework (with a minor modification needed for one thread – one client behaviour that has just been included in the "final" release of the framework);
- JMiramare replication algorithm (with the server-side load balancing disabled).

The test application which was constructed to allow this evaluation was a simple Axis web-service, those state is an array of Integer values.

The size of the array was set with a call to the init() method of the service. The test methods exposed by the service were:

- increment() – increments each array member by one, returns no value;
- read() – returns mean value of the array rounded to an integer;
- incrementAndRead() – combination of the above two methods;
- readAll() – returns the entire array.

The client application part of the evaluation was run on a multi-node system. The "master" node distributed the load (number of threads that each node should run) and values of configurable parameters to the other nodes and coordinated the whole system via TCP. At the end of the test, the "master" collected and sumed the data from all the nodes and stored it to a file. Up to five client nodes were used to inject the load into the system.

Each node started the assigned number of threads that acted as actual "clients" for the web service. The threads were started one at a time at one second intervals. Once all the threads had started, the warm-up period began, when threads (clients) make calls to the cluster, but the data was not collected. The actual test was run at the end of the warm-up period. During "runtime" both successful and failed calls were counted and the latency of successful calls was measured. Each client slept for a short period between calls.

The following parameters were used for the test:

- Tests were performed with three different array sizes (1, 100 and 1000.)
- For each object size, tests were run separately for each of the four methods. (Except for readAll method, tested for array size 100 only.)
- Both "warm-up" and "runtime" intervals were 10 minutes long.
- Clients slept for 600 ms between calls.
- From the collected data we calculated throughput (operations/second) and mean-latency (milliseconds) as observed by the clients.
- For comparison, we tested JBoss with the Adapt-framework configuration (Array size 1, 100, 1000) and JBoss-only configuration (Array size 100) as well.

The test results that were obtained will now be described - only the clients/throughput graphs are shown.

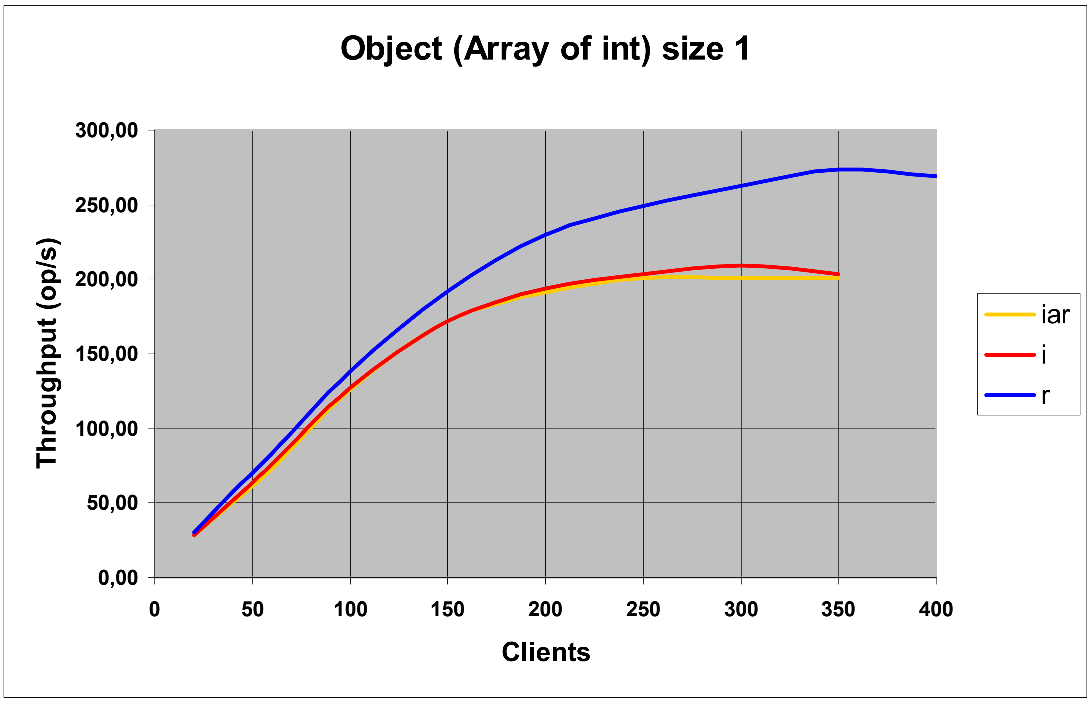


Figure 1, Adapt-framework with JMiramare.

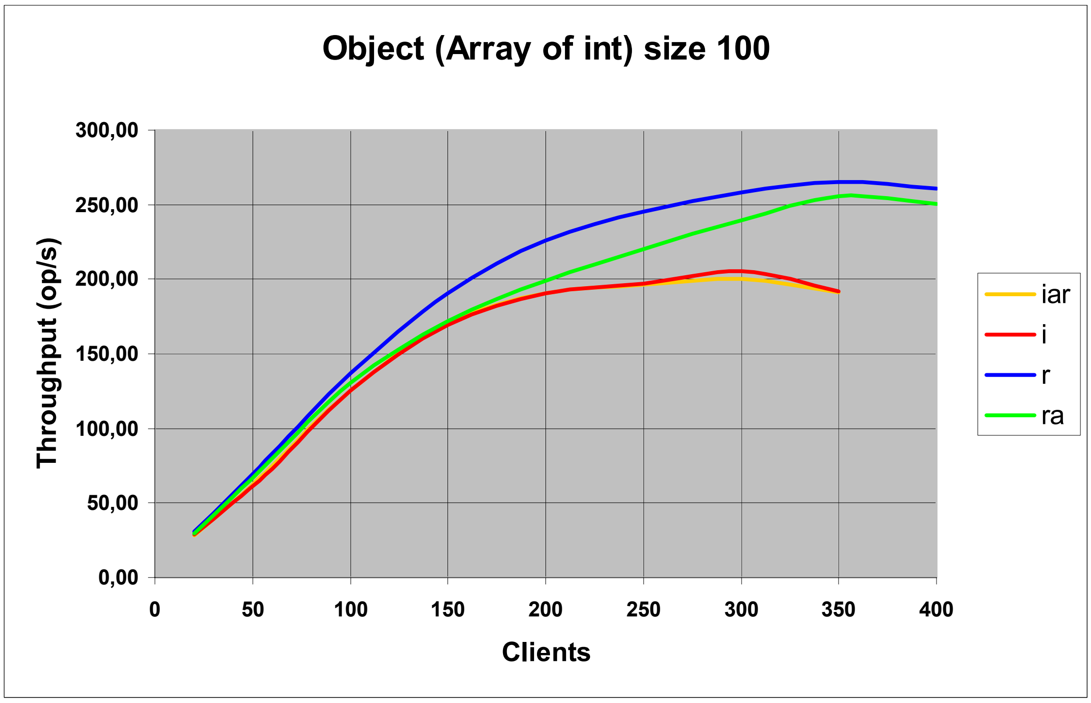


Figure 2, Adapt-framework with JMiramare.

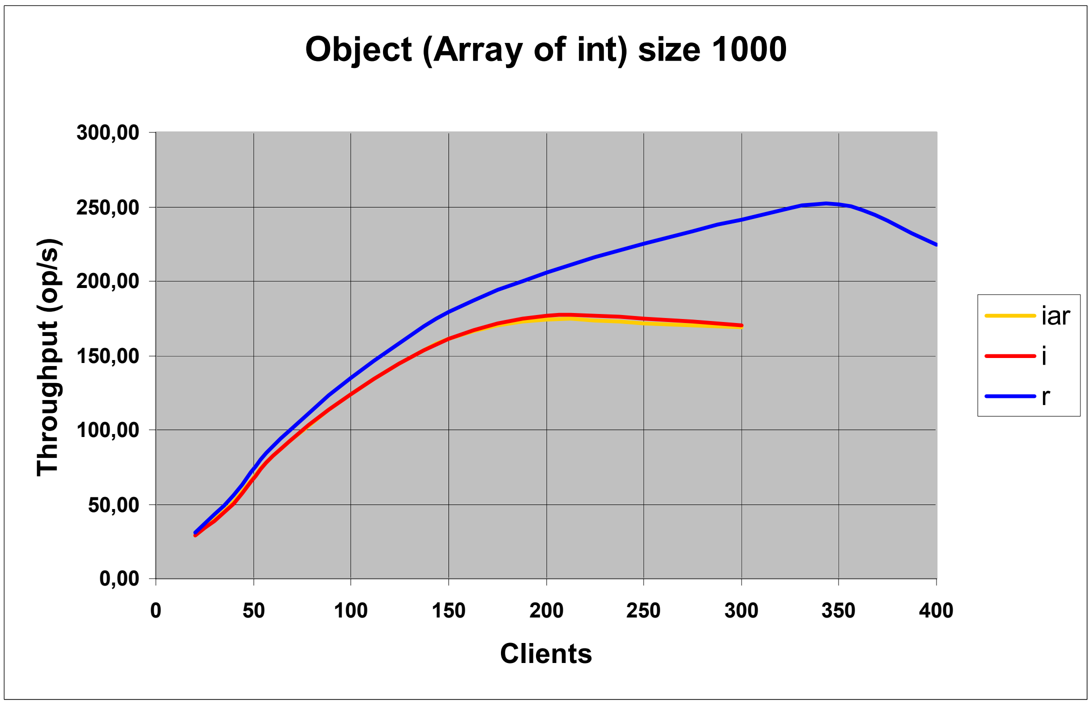


Figure 3, Adapt-framework with JMiramare.

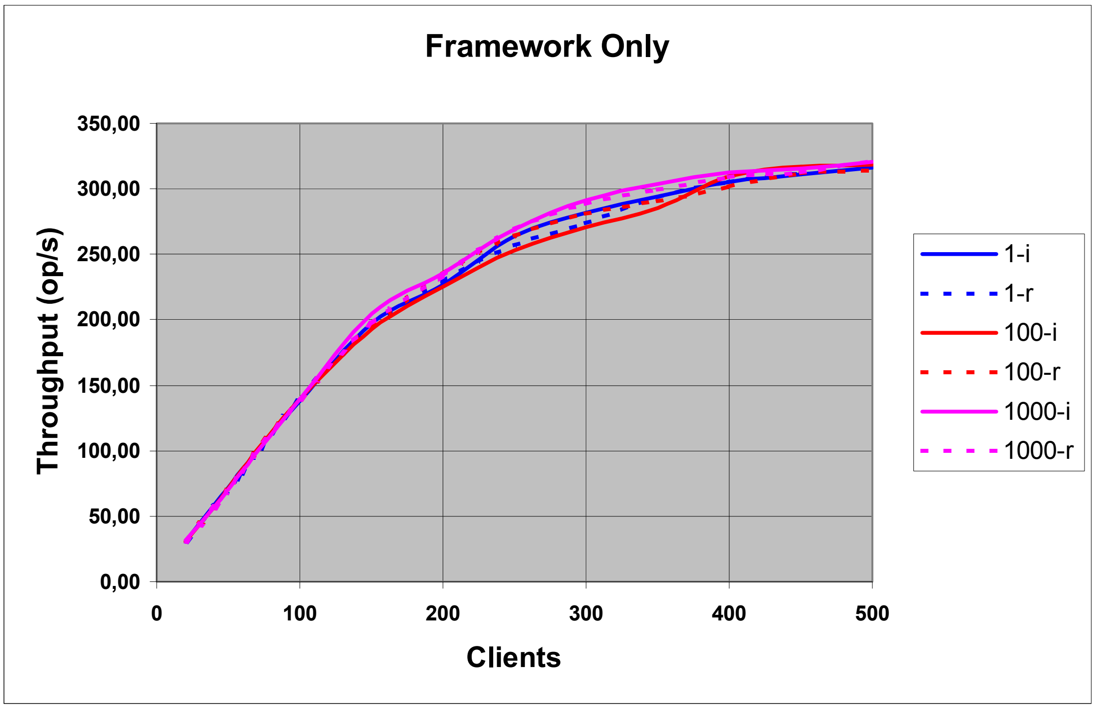


Figure 4, Adapt-framework only.

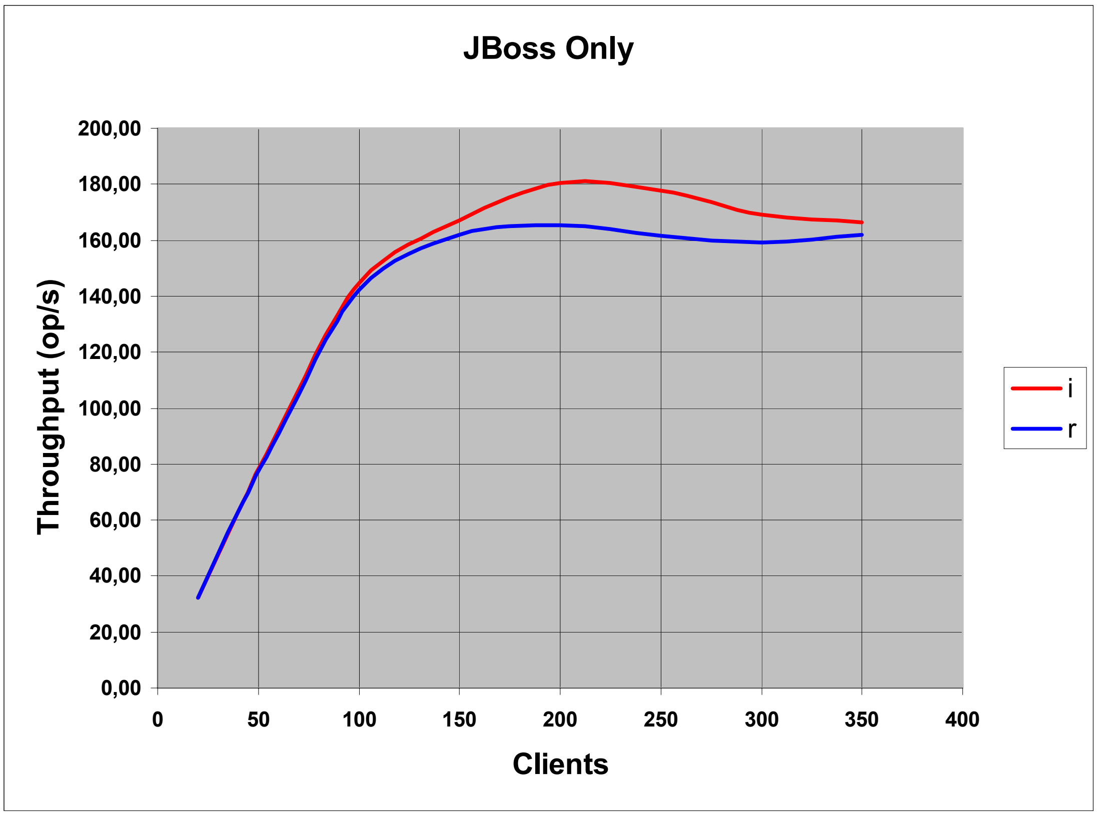


Figure 5, JBoss only.

The service latency was measured on the client side, for a single client. For the replicated case, the cluster was set up like in the above tests, while for the non-replicated (JBoss-only) case, a single server machine was used.

Array Size	*Method*	*Mean Latency Jmiramare*	*Mean Latency Framework*	*Mean Latency JBoss-only*
1	iar	76,20	22.92	21,10
	i	75,68	20.51	20,11
	r	29,33	20.96	18,90
	ra	29,60	20.92	21,55
100	iar	73,43	23.23	18,29
	i	82,43	21.38	19,75
	r	28,62	22.59	20,12
	ra	36,32	25.51	34,38
1000	iar	90,83	23.95	19,44
	i	79,15	23.85	17,53
	r	27,99	25.03	19,62
	ra	107,74	83.41	228,97

The evaluation of JMiramare led to the following conclusions and observations:

- Since we were unable to come up with a satisfactory server-side load balancing mechanism, one that would converge in all test cases, we distributed the load evenly among the cluster members from the client side. (The primary goal of these tests was to measure JMiramare and the replication-framework performance, not the efficiency of a load balancing algorithm.)

- The SOAP parsing of the replies on the client side consumed a considerable amount of the CPU time which made the introduction of a pause between the calls absolutely necessary. Still, the results of the readAll() method were compromised because the clients' CPUs saturated long before the servers' did.

- In the JMiramare replication algorithm, the read-only operations did not require multicasts because the state of the service object remained unchanged, so the higher throughput was expected.

- The size of a JMiramare multicast message varied with object (array) size:

Array size	*Object state (bytes)*	*Multicast msg (bytes)*
1	296	1307
100	692	1703
1000	4292	5304

- The Adapt-framework's throughput (with no replication algorithm plugged in) in the above tests was limited by the clients' CPU usage. Even with 500 clients, the server CPU usage was still at around 70% on each cluster member. The results were significantly better than in the JMiramare case for the update operations. As expected, for the read-only operation that did not require a multicast in the JMiramare, the results were quite similar for the two configurations.

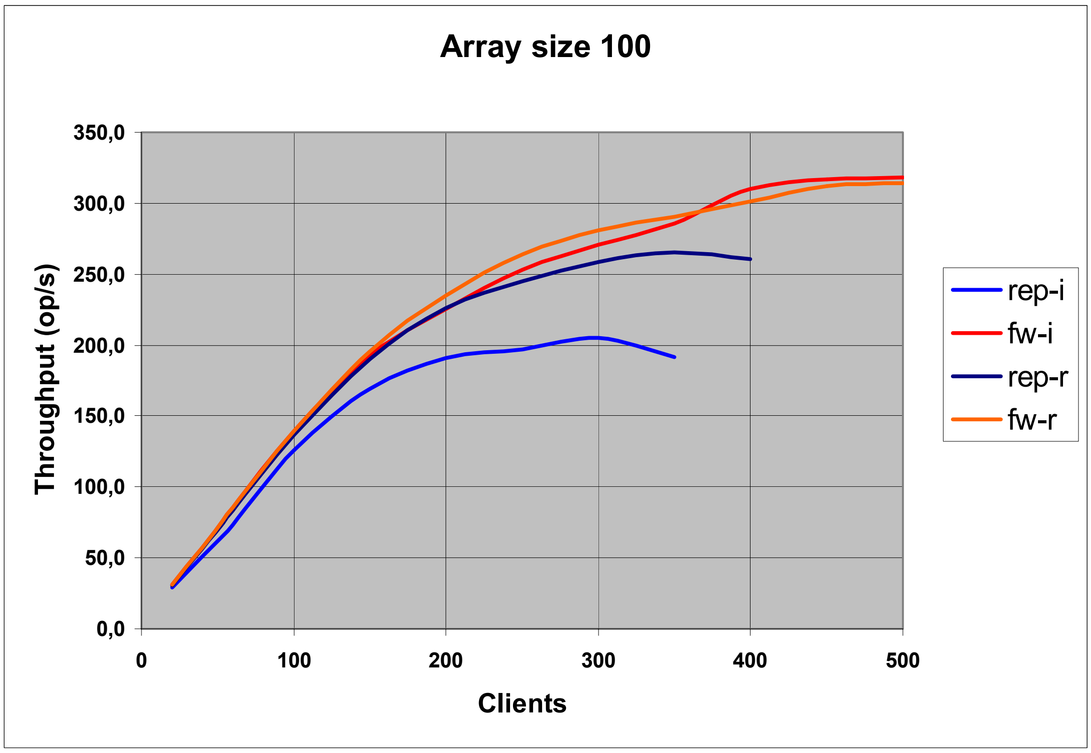


Figure 6, Adapt-framework with JMiramare vs. Framework-only.

- The JBoss-only test results remain hard to explain. That is, the configuration from which we expected the best results (highest throughput) actually produced quite the opposite. The cause is most likely to have been a configuration error although we did not manage to discover it. The only other explanation that would explain this behavior was that the Adapt-framework possessed a custom session handling implemented.

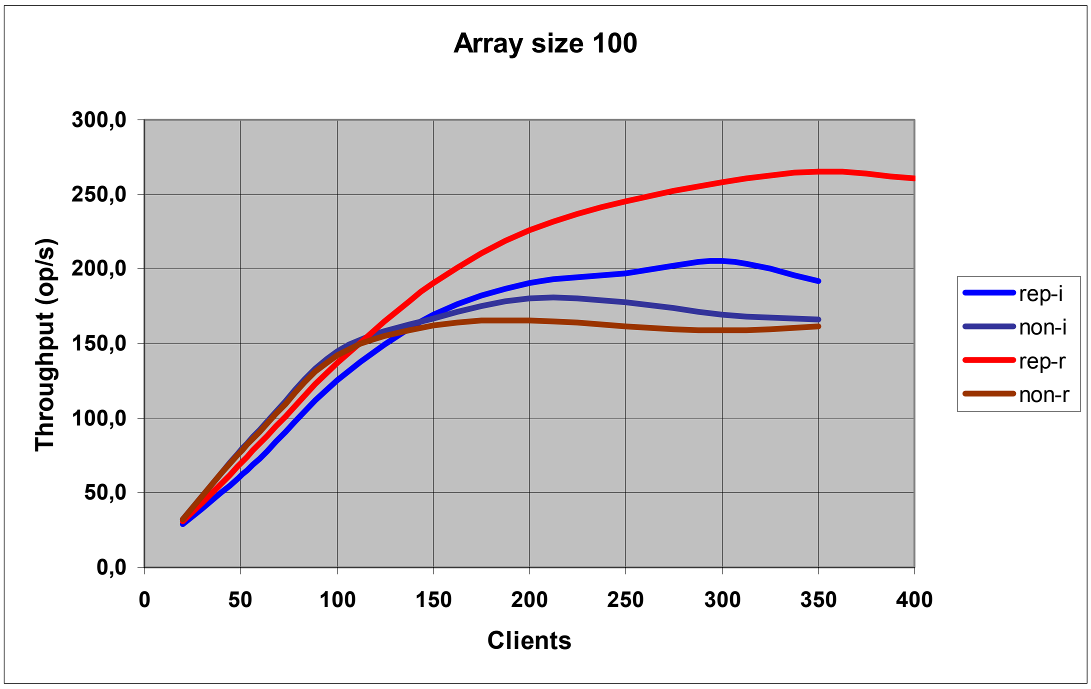


Figure 7, Adapt-framework with JMiramare vs. JBoss-only.

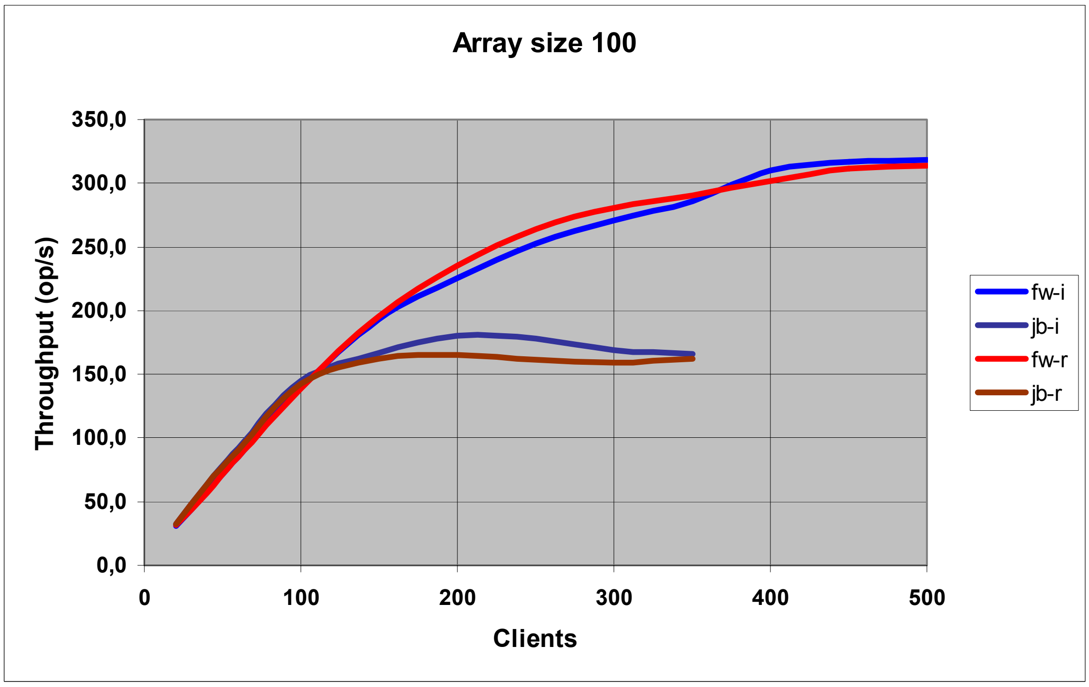


Figure 8, Adapt-framework vs. JBoss-only

1.1.1.2 Via the Middle Tier

The evolution of the Middle Tier of the Basic Service support has been divided into two part, the evaluation of the framework and replication algorithm.

Framework evaluation

To evaluate the performance of the framework, we used the ECperf benchmark provided by Sun. The benchmark uses a standard manufacturing / supply chain /

inventory problem as an example to exercise the numerous features defined in the J2EE specification. The benchmark consists, for one part, of several bean archives to be deployed on the application server and a database schema to be instantiated and populated, and, for the other part, of a configurable client application that uses those previously deployed beans to stress the server. The main configuration variable is the target transaction rate at which the client will try to make the server operate by creating an appropriate number of independent client threads. Once the test is completed, the client software collects statistics from the different threads and extracts an average "business operations per minute" computed over the steady state phase of the run (the ramp-up and ramp-down phases are ignored).

In our experimental setup, the JBoss application server and the PostgreSQL database management system ran on a single 2.4~GHz Pentium~IV Linux machine with 1~GB of RAM (which was otherwise idle). Clients were executed on a different machine, located on the same local area network.

In this environment, we measured the performance of two configurations. The first was JBoss, without the ADAPT framework (and without the JBoss replication mechanism). The second was the ADAPT modified configuration, with a "dummy" replication algorithm that caused it to behave as a normal non-replicated server. The results of this experiment are shown in Figure 9.

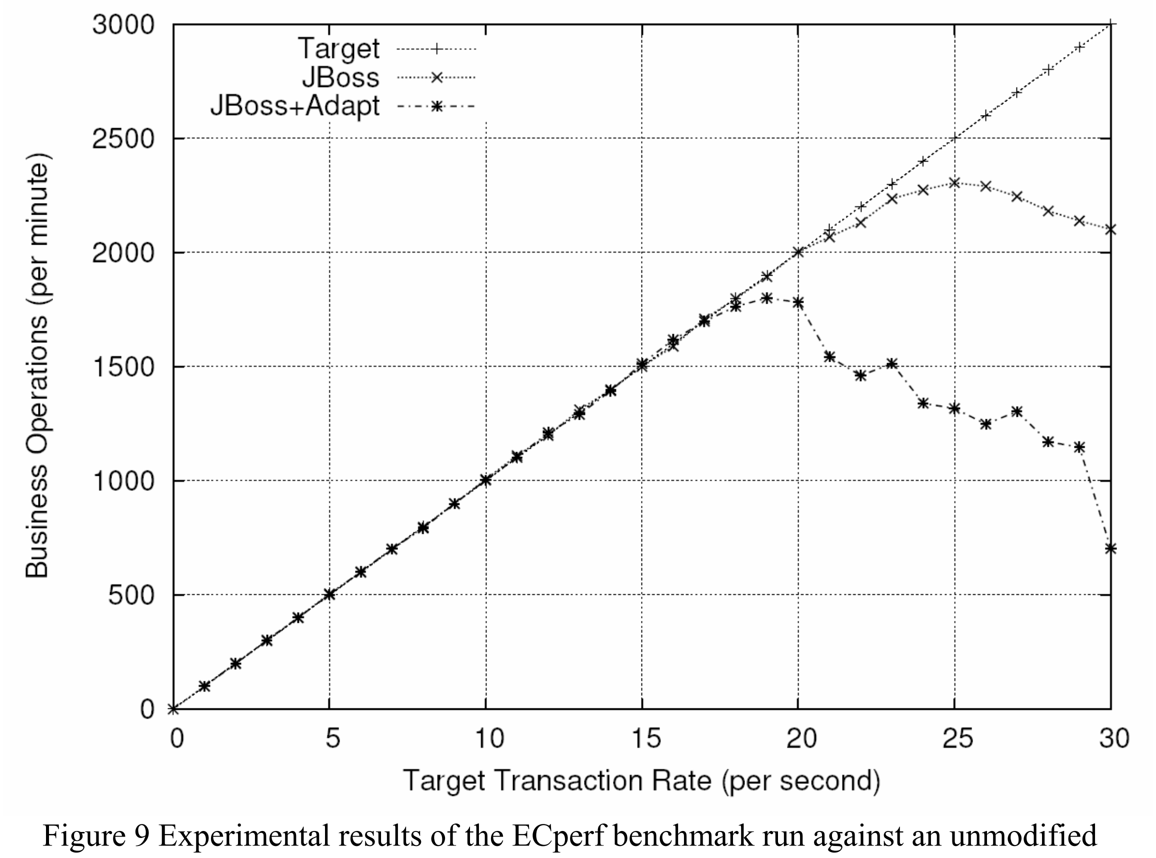


Figure 9 Experimental results of the ECperf benchmark run against an unmodified version of JBoss and against the ADAPT framework. The target line shows expected throughput according to the chosen transaction rate.

The general shape of both curves was the same: the number of operations increased linearly with the target transaction rate until the server became overloaded. After this point, some transactions began being aborted, due to timeout or contention on the database management system. This overload landmark was reached at 21 transactions per second for unmodified JBoss, and at 18 transactions per second for the ADAPT configuration. The throughput maxima were 2105 and 1800 business operations per minute respectively. In other words, the penalty for adding the framework was a 22% drop in throughput. One should also note that beyond the overload landmark, performance degraded more steeply in the ADAPT configuration.

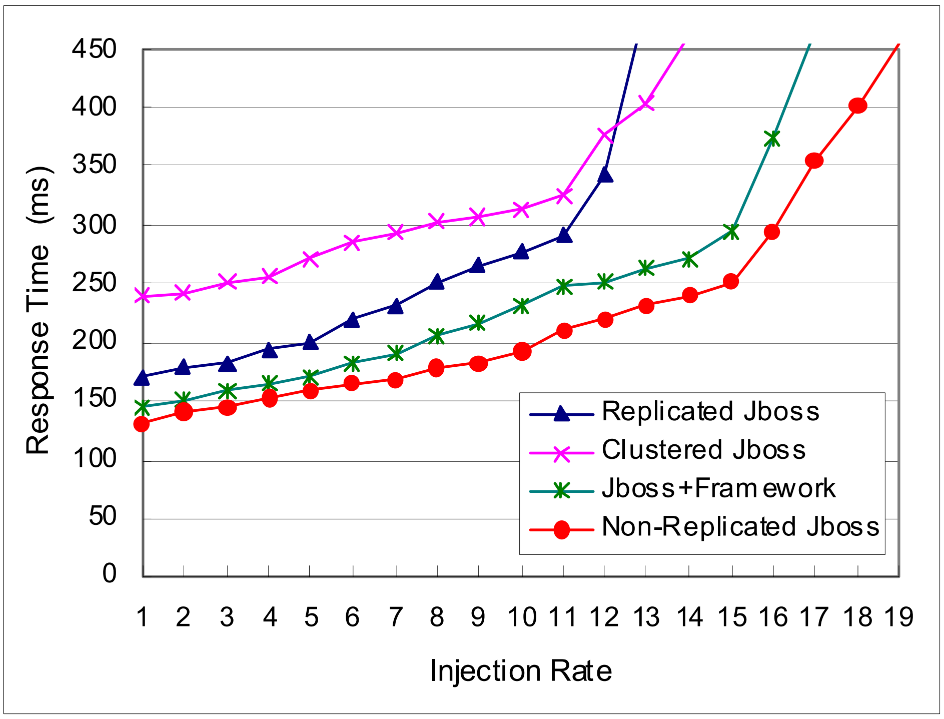
Clearly, adding the framework had a real impact on the server. However, at 1800 ECperf business operations per minute, the performance was still quite practical, enabling developers to run realistic benchmarks against their prototype replication algorithms.

Replication algorithm evaluation

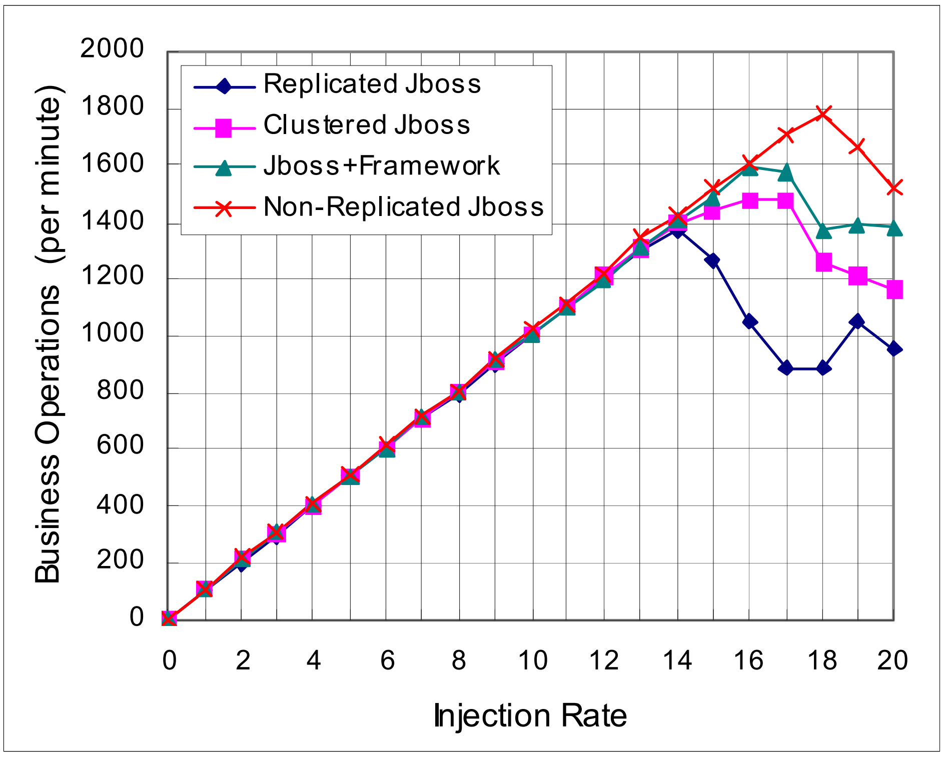
The evaluation of the middle tier of the ADAPT Basic Service platform also involved analyzing the performance of the SIB system which is the replication algorithm for the application server based on the framework. We used three suites of experiments to evaluate SIB system. First, we used the ECperf benchmark to evaluate the performance on a "real" application and compare it with JBoss's existing replication technique. A second test suite was used to present a series of micro benchmarks that show the performance for different components and database access patterns. The third experiment evaluated failover. All tests were run on a cluster of PCs (3.0 GHz Pentium 4 with 1 GB of RAM) running RedHat Linux. The configuration consisted of one machine emulating clients, one machine running the web server (if needed), two machines running JBoss application server 3.2.3 instance, and one machine running DB2 as our database system.

ECperf emulates businesses involved in manufacturing, supply chain and order/inventory management. The application is split into four domains: customer, manufacturing, supplier domain, and corporate. The main configuration variable is the *transaction injection rate* (IR) which refers to the rate at which a specified subset of business transaction requests is injected into the system.

In this experiment, we evaluate the following architectures. (1) A regular, non-replicated JBoss server as baseline for comparison. (2) The JBoss server including the framework without the replication algorithm to evaluate the overhead of an abstraction layer useful for reusability and platform independence. (3) Two application server replicas using our eager replication algorithm. (4) Two application server replicas using JBoss's own replication solution, which we refer to as JBoss clustering. For both (3) and (4) we did not take advantage of load balancing, and submitted all requests to one server. JBoss clustering uses passive, warm, and eager replication. If a client request triggers execution on several stateful components, state transfer takes place individually once execution on the component has terminated. Although eager, problems occur if state propagation for some of the components was successful but the primary fails before committing the transaction at the database. In this case the backups have a partially replicated state while the database transaction aborted. Hence, JBoss clustering does not provide state consistency or exactly-once execution.



(a) Response Time



(b) Throughput

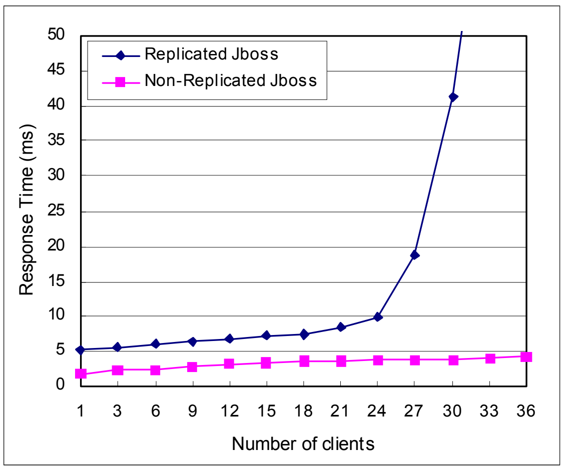
Figure 10, ECPerf Comparison

Figure 10 shows the results of the experiment measured over the steady state phase of the run (the ramp-up and ramp-down phases are ignored). The figure (a) shows the average response time for order entry transactions of the customer domain. At low throughput, the framework adds around 10 ms to the non-replicated JBoss, our algorithm (indicated as Replicated JBoss) adds 25 ms while JBoss clustering adds around 100 ms. This gives an overhead of around 25% for our algorithm plus the framework, and 70% for JBoss clustering. The latter performs so badly because it sends state after each method invocation while our solution only communicates at the end of the transaction. As a comparison, Moser et. al. in [A fault tolerance framework for CORBA. In Proc. of the Int. Symp. on Fault-Tolerant Computing, Washington, DC, USA, 1999.] indicate around 15% overhead for FT-CORBA (primary-backup) compared to non-replicated CORBA. With increasing IR, the response time in all systems increases steadily until saturation point. The gap between non-replicated JBoss and the eager algorithm increases slightly but steadily, while it remains nearly the same for JBoss clustering until around 11 IR beyond which it becomes significantly worse. More information about the saturation point can be found in figure (b). This figure uses the average *business operations per minute* to represent the maximum achievable throughput when the IR increases. The maximum in each curve shows the system shortly before saturation. Our algorithm leads to saturation at an IR of 15 due to CPU overhead. JBoss clustering saturates at 17 IR (also due to CPU) while the non-replicated JBoss saturates at 19 IR. In this case, our DB2 server is the bottleneck although the CPU was also already quite heavily loaded. By optimizing the DB2 configuration (we used the default configuration of DB2 with only small adjustments), the CPU might become the limiting resource also in this case. The reason for earlier saturation of our approach compared to JBoss clustering is higher CPU overhead (keeping old responses to guarantee exactly-once, keeping information during transaction execution to send all state in a single message).

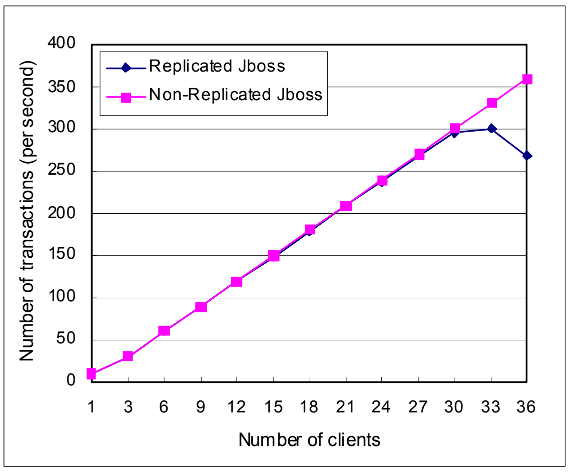
In summary, we believe that our approach provides acceptable performance considering the strong consistency guarantees that it provides. It compares favorable with JBoss's clustering mechanism. Nevertheless, the overhead is not negligible. We believe, however, that more “engineering” work in optimizing our in-memory data structures could lead to further improvement. In order to better understand where to start such optimizations, the next section presents results on simpler applications in order to detect potential bottlenecks.

In our second experiment suite, we evaluated the overhead of replication for different components and component combinations. We considered the following cases:

- Test 1: No database access takes place.
- Test 2: Database access (update) takes place but no conflicts occur at the database. That is, different clients access different tuples.
- Test 3: Database access takes place and all transactions conflict. That is all requests access the same tuple.



(a) Response Time



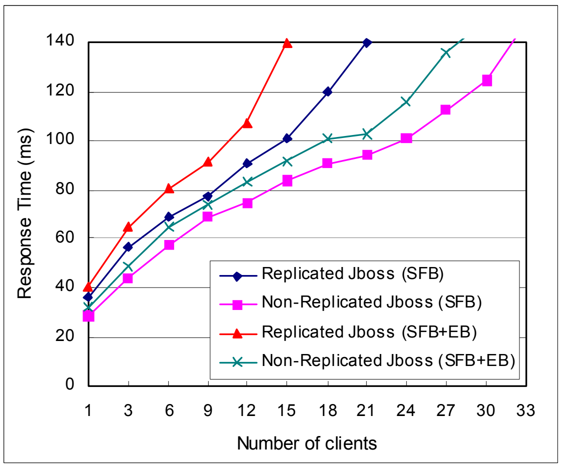
(b) Throughput

Figure 11, Component Analysis for No database Access

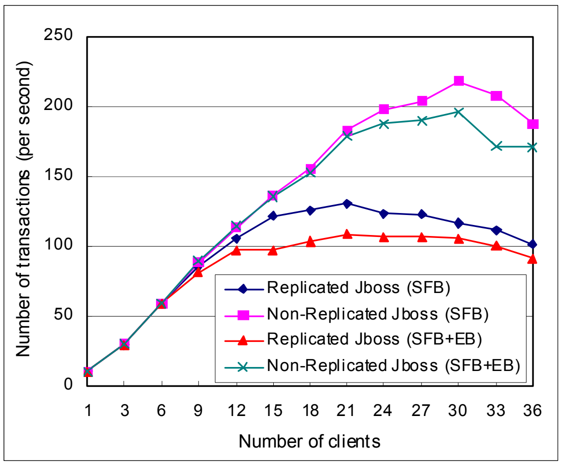
In Test 1, a request triggers the execution of a single method of an SFSB. Tests 2 and 3 have two different versions. In the first, a request executes only on one SFSB which makes the database call. In the second, a request calls a SFSB, which calls an EB to access the database.

The main configuration variable is the number of clients. Each client is configured to submit 10 requests per second. However, since a client does not submit a new request before it receives the response for the previous request, if the execution time is longer than 100 ms, the real injection rate is smaller than 10/sec.

Test1: No database access. Figure 11 shows (a) the average response time and (b) the throughput achievable with increasing number of clients. Response times increase slowly for both the replicated and non-replicated system. Below the saturation point, the replication algorithm (including the framework) has an overhead of around 4 ms. This is very low in total numbers, but means an overhead of around 100% for medium number of clients since response times are generally very small. This is the worst case scenario for our algorithm since it contains only SFSBs which all must be replicated. At 24 clients, response times increase sharply due to CPU saturation, and the final saturation is after 33 clients. The non-replicated system does only saturate at around 66 clients again due to CPU overhead. Since the system is CPU bound, and the non-replicated system takes half the time to execute one request compared to the replicated system, it can execute double as many requests before saturation. There are two solutions to improve the results of the replicated system. The first is to improve the implementation of the algorithm (e.g., data structures, access paths). This, however, can only succeed to a certain point. After that, alternative replication strategies have to be found, e.g., lazy replication.



(a) Response Time



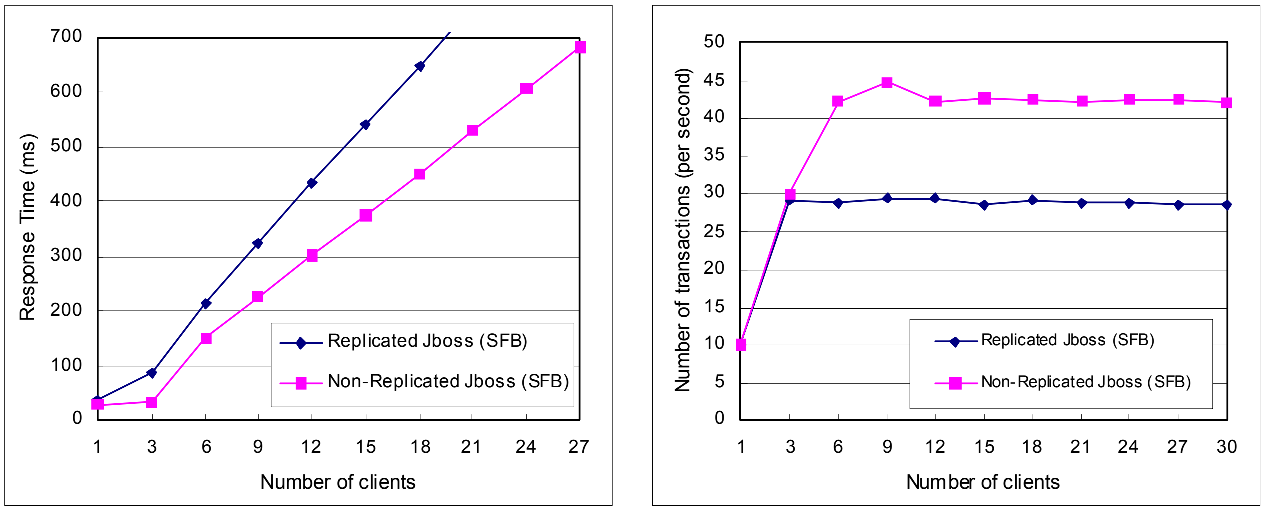
(b) Throughput

Figure 12, Component Analysis for conflict-free database access

Test 2: Conflict-free database access. Figure 12 shows the results of test 2, in which transactions access the database but concurrent transactions never conflict. The figure contains graphs both for the SFSB only and SFSB/EB combinations. Let's first have a look at the SFSB only case. Compared to the figure (a) for no database access, response times increase more steeply, and are generally higher. This is due to the database access. When the number of clients is smaller than 15, the overhead of the replication algorithm is stable at around 15 ms. The total time spent in the replication algorithm is higher than with no database access (4 ms) because the marker has to be inserted into the database (if a transaction does not update the database, no marker is inserted). In this scenario, 15 ms only means an overhead of 20% for medium client numbers since transaction execution is generally long. With more than 15 clients, the time spent in the replication algorithm increases linearly with the number of clients and the throughput increases only very slowly. At 15 clients, the CPU overhead is around 85%. After that, it does not increase fast because the system always waits for operations at the database to complete. The saturation point is at 22 clients. The non-replicated server reaches saturation with 33 clients.

When database access is filtered through EBs, response times both for the non-replicated and the replicated system are generally higher due to the EB overhead. However, the relative performance is similar to the SFSB only case.

The conclusion is the easy observation that if the original system has high execution times, then the overhead of the replication algorithm has not such a big relative effect than with small execution times.



(a) Response Time (b) Throughput

Figure 13, Component Analysis for conflicting database access

Test 3: Conflicting database access. Figure 13 shows the results when all transactions conflict at the database. We only present the SFSB only case, since the effect of using EBs is similar to test 2. Generally, response times of this test are much larger than in test 2 due to the long blocking times at the database. They increase sharply with the number of clients for both replicated and non-replicated case. The difference between replicated and non-replicated system is bigger than in test 2 and also increases faster than in test 2. The reason is that the replication algorithm generally increases the execution time for each transaction. Assume transaction T1 holds a lock and T2 and T3 wait for the lock. The time T1 needs longer to finish due to replication is also added to T2's and T3's execution time. Additionally, the longer execution time of T2 is added to T3's execution time. This means, waiting times are cumulative. We can also see that the maximum throughput in this test is only around 1/4 of the one in test 2 for both the replicated and non-replicated system due to the blocking.

As a conclusion, although the CPU is not saturated, the CPU overhead of replication limits its performance. Although the response time increase is due to longer waiting times at the database, it is caused by the computation overhead.

1.1.1.3 Via the Back Tier

An analysis of the scalability and overall performance of the Middle-R system was undertaken by varying the number of sites and the load (see Appendix 1). This analysis was performed on a cluster of 15 *SUN Ultra-5_10* workstations connected to a fast Ethernet network. The analysis consisted of performing the desired transaction workloads, on the Middle-R system, where payloads were carefully constructed to have the desired ratio of update to query transactions, with the queries ranging from accessing the entire table to accessing single data items.

Six sets of experiments were conducted. In the first four experiments each transaction only accessed a single basic conflict class and the experiments focused on issues like scalability, response time behavior, and communication overhead. Experiment five analysed the behavior of the system when transactions accessed more than one basic conflict class. The first five experiments used the REORDERING algorithm. The last experiment compared abort rates of both NODO and REORDERING.

The following conclusions were drawn:

- When compared with a standard distributed locking, that does not scale at all, the Middle-R system proved that it is possible to scale by combining locking with a group communication based solution.
- Middle-R showed that it is possible to scale with high ratios of updates unlike other middleware-based approaches in which there was little scalability.
- Middle-R showed that by exporting writeset acquisition and setting services it becomes possible to attain scalable replication at the middleware level, whilst decoupling the replication logic from the regular transaction processing logic.
- The scalability gains were both in terms of throughput and latency. That is, throughput was not improved by trading off latency.
- The communication overhead of Middle-R was analysed and it was found that communication was not a bottleneck to performance.

1.1.1.4 Highly available advanced transaction support

Long running activities cannot rely on ACID transactions since their long duration and are typically based on advanced transaction support. However, their long duration becomes an issue in terms of availability. A replication algorithm within the adapt framework has been developed for long running activities that exhibits high availability, that is, in which in the advent of failures, long running activities are not aborted and recreated upon failover. The performance of this support for long running activities has been evaluated and compared to the non-replicated case. The used replication model differs from the one used for ACID based transactions and resorts to vertical replication, in which pairs of application servers and databases are replicated.

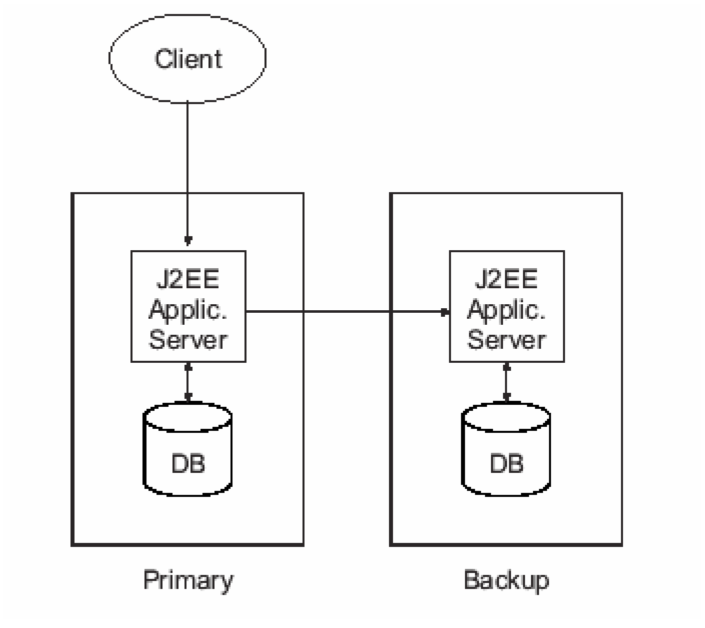


Figure 14.

Since there are no existing benchmarks for advanced transactions, a specific benchmark was developed based on an e-shop. The below figure shows the structure of the advanced transactions. More details are provided in Appendix 11.

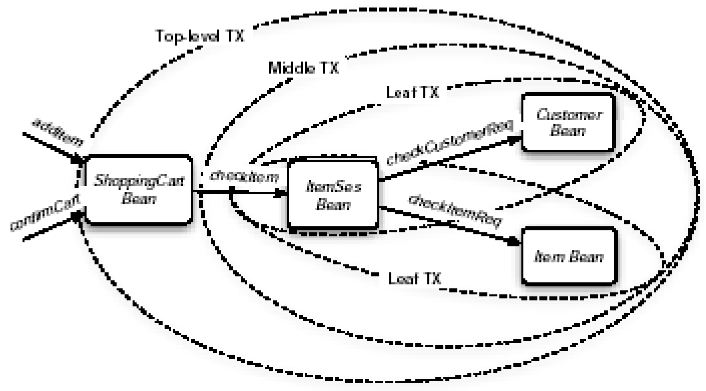


Figure 15.

The benchmark was instrumented by resorting to the ECPerf client and adjusting it to the target application. The results of the benchmark showed that the replication protocol had a reasonable overhead when compared to the performance of the non-replicated execution. In the below figure, it can be seen the performance of the approach in terms of latency and throughput.

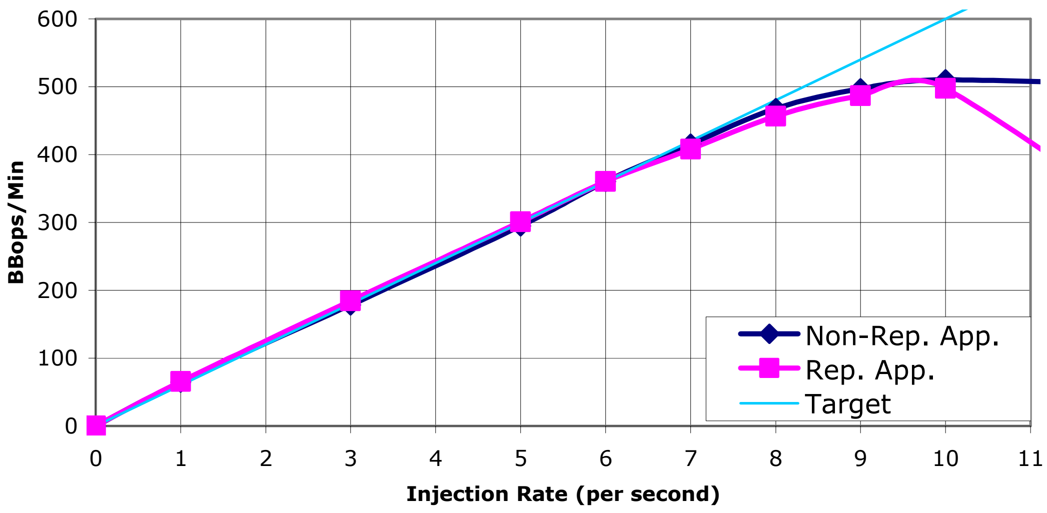


Figure 16.

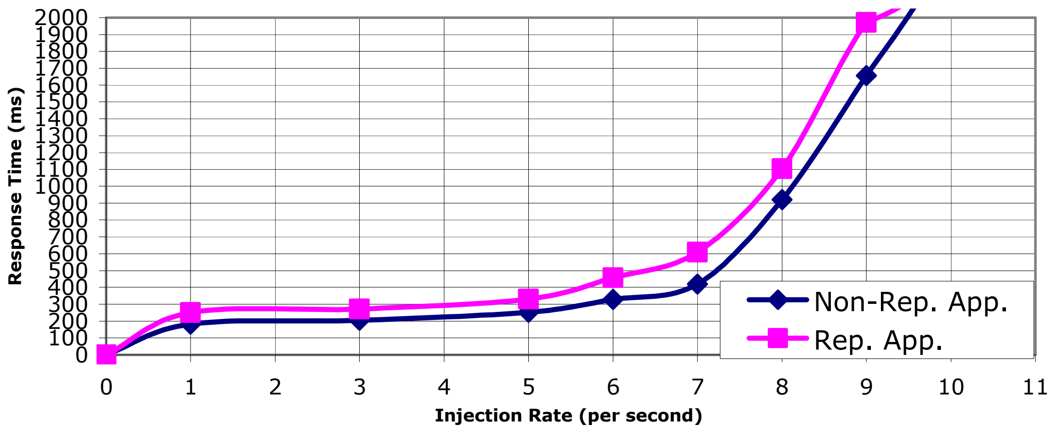


Figure 17.

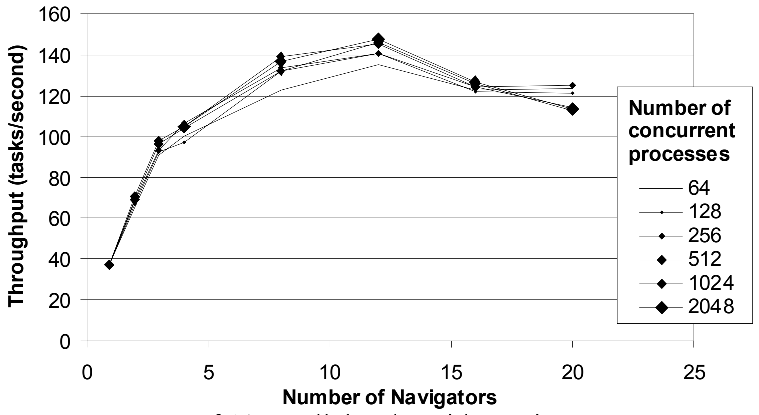
1.1.2 Composite Service Support Experiments

The performance, scalability and adaptability of the composite service execution engine were determined by performing a series of experiments. These experiments used the following set of process definitions that exercised differing characteristics of the engine.

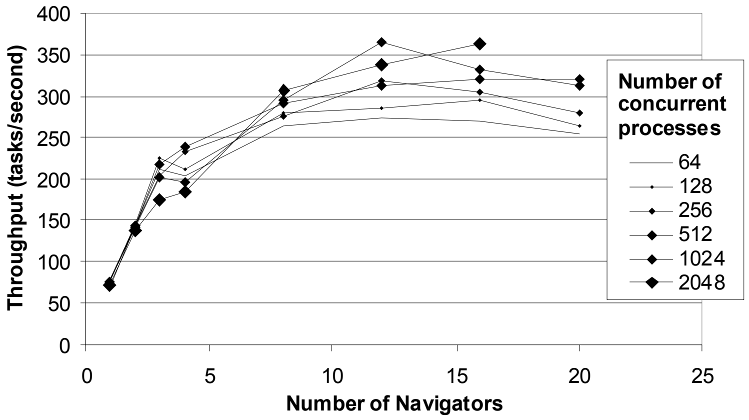
- "A long chain" of serial executed empty (null operation) tasks were used to determine the coordination latency of the coordination engine (Volatile Storage). The results from this experiment were as follows:

Sequence Length (tasks)	1	5	10	15	20	25
Execution Time (s)	< 0.001	0.005	0.011	0.016	0.027	0.032

- "Single empty (null operation) tasks" were used to determine the average throughput and average latency involved in initiating a task. The results of this experiment are presented in the following two graphs. In the Figures below, the "navigator" should be interpreted as the number of replicas of the engine as it runs distributed on a cluster.



Processes of 10 parallel tasks with persistent storage



Processes of 10 parallel tasks with volatile storage

Figure 18, Scalable execution: average throughput of the system using an increasingly large number of parallel navigators running tasks lasting 0 seconds

- "Single long idea task" were used to determine the scalability of the engine with respect to coordinating large numbers of tasks. The results of this experiment are presented in the following graphs.

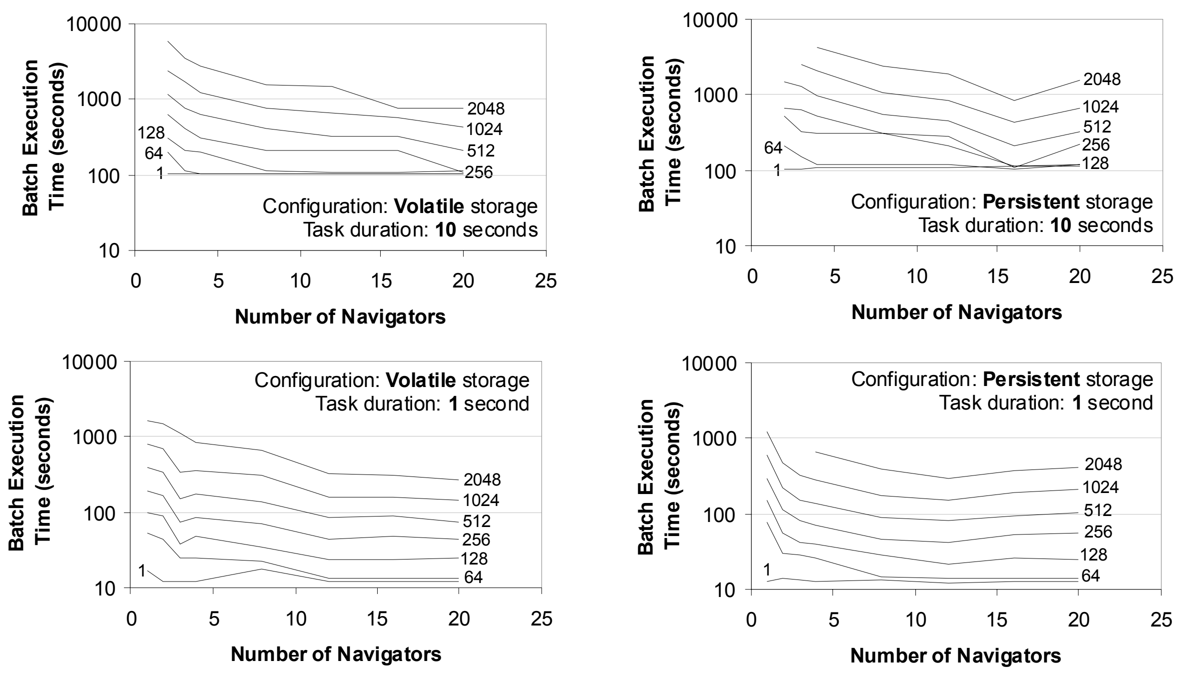


Figure 19, Scalable Execution: Execution time of batches processes having 10 tasks and sequential control flow topology as a function of 4 variables: 1) the number of navigators (X-Axis), the workload size (Z-Axis), the duration of the tasks and the system configuration (volatile or persistent storage)

1.2 Strategy for Evaluating Functional Capabilities and Attributes

The evaluation of the main functional goals of the ADAPT project has been done in the context of two complementary approaches: firstly the construction of a proof-of-concept application [1][2] on the ADAPT platform and secondly the analyses of the achievements of the project in the areas that were key technical goals.

1.2.1 Proof-of-concept implementation

Whilst the ADAPT platform was always intended to be for “programming in the large”, the construction of a very large application to test “programming in the large” with the ADAPT platform was always beyond the financial resource of the project. So, the proof-of-concept implementation was based on the implementation of the WS-I Sample Applications, which in turn is based on a supply chain management scenario. This application involves a customer system (referred to as the demo system) making requests on a retailer system, which in turn makes requests on a set of manufacturing systems. The interactions between these systems are performed by web service invocations. The interaction patterns range from simple synchronous and asynchronous invocations to callbacks, providing an interesting set of scenarios.

However, the construction of this application confirmed the observation that the ADAPT platform is not suitable for “programming in the small”. Despite the fact that the application consisted of about 20 persistent entities, 24 basic services and 6 composite services, it was not large enough to meaningfully utilize the ADAPT platform. Because of this, the applications’ components had to be implemented at a level of granularity that was one level higher than would normally have been appropriate. Hence, what would have been basic services became composite services and objects became basic services. This resulted in a very elaborate design that was not merited by the complexity of the application. The initial cause of this was the original design decision that basic services would not be permitted to invoke other services. Whilst this greatly simplified the implementation of the basic service it was an onerous restriction for the application designer. The original design decision also resulted in the creation of a large number of basic services.

The two central goals of the web service replication were to provide consistent invocation failover, and management of the session state between the replicas. It turned out that this application did not naturally have any session state that needed to be maintained. Interactions-based web-pages are often structured as a series of steps, one following on from the next. In such an interaction a session state is very useful for maintaining the state of the interactions. But for the web service based interaction, the nature of the interaction between the “invoker” and “service” is more complicated, the main reason being that the “invoker” could be multi-threaded, these threads possibly being caused by invocation on the “invoker”. The result is that the series of invocations from the “invoker” could be unrelated, and hence no session state is appropriate, not even a security session. For such interactions a context, either explicit or implicit, would be more appropriate. For the demonstrator application the context, such that it was, consisted of explicit parameters to invocations, that mapped directly to database entries. So, whilst it could have been possible to build an application that did rely on session state, this application did not.

To evaluate the service description capacities of the ADAPT platform, the interaction specified within the WS-I Sample Applications were formulated using the ADAPT interaction constraints language. The resulting definitions were comparatively concise and easy to construct. This could be a result of the interaction within the WS-I Sample Application being inline with the type of interaction that was envisioned when the interaction constraints language was designed. A fragment of an interaction constraints specification, from the WS-I Sample Application, is given below:

```
<ic:invokeOutput operation="poSubmit" portType="manufacturerCallPort"
    participant="Manufacturer" participantPortType="manufacturerPort">
    <ic:choice>
       <ic:sequence>
           <ic:invokeInput/>
               <ic:serviceInput operation="snSubmit" portType="warehouseCallbackPort"
                   participant="Manufacturer"
                   participantPortType="warehouseCallingbackPort">
                   <ic:choice>
                        <ic:serviceOutput/>
                        <ic:serviceFault name="CallbackFault"/>
                        <ic:serviceFault name="ConfigurationFault"/>
                    </ic:choice>
                </ic:serviceInput>
            </ic:sequence>
            <ic:sequence>
                <ic:choice>
                    <ic:invokeFault name="POFault"/>
                    <ic:invokeFault name="ConfigurationFault"/>
                </ic:choice>
                <ic:serviceInput operation="processPOFault"
                    portType="warehouseCallbackPort" participant="Manufacturer"
                    participantPortType="warehouseCallingbackPort">
                    <ic:choice>
                        <ic:serviceOutput/>
                        <ic:serviceFault name="CallbackFault"/>
                        <ic:serviceFault name="ConfigurationFault"/>
                    </ic:choice>
                </ic:serviceInput>
            </ic:sequence>
        </ic:choice>
    </ic:invokeOutput>
</ic:protocolType>
```

Process description capabilities of the ADAPT platform proved to be capable of specifying the composite service that was required by the demonstrator. This is remarkable given that the designers of the WS-I Sample Application did not originally intend parts of the application to be implemented as a workflow system managed process. The only pragmatic concession that was made to the composite service design was that some additional operations were added to some basic services to create and transform some of the messages types. This could have been done within the ADAPT process definition language, but would have been cumbersome, because some of the messages contained a large number of fields.

1.2.2 Service description goal

The service description goal of the ADAPT project was to investigate supplementing the usual interface description of services, with additional information that could be used to make the design and management of the service easier. The ADAPT project produced three main results in this area: (i) interaction constraints language (ii) transactional semantics, which both aid in the design phase and (iii) performance properties, which aid at runtime.

The transactional semantics allowed the macro behavior of operation invocations to be specified, for example, if an operation had a corresponding compensation operation or if the invocation of the operation implied that a 2-phase commit protocol would need to be performed later. The transactional semantics specification consisted of a fairly comprehensive set of possibilities, but the ADAPT project didn't specify any new transaction models or investigate how operations with different transactional semantics could be composed.

The interaction constraints allowed the context, within which it is appropriate for an operation to be invoked, to be specified. For example, if a *submitOrder* operation was expected only to follow a *getCatalog* operation, this can be specified. The initial goal was to specify an interaction constraints language in XML which was expressive enough to describe more interactions. Judging by our experience specifying the WS-I Sample Application interactions, this appears to have been successful. Following the specification of the interaction constraints language its use to analysis composite services was investigated. It became clear that the language could be mapped to PI-Calculus and Promela. This allowed tools such as Spin to be used to analyse the composition of services, identifying incompatibilities in the requirements of the component services.

The area of interaction constraints has over the last three years become an area of considerable interest to industry and standards bodies. Shortly after the start of the ADAPT project the OASIS standards body chartered the WS-CDL working group to produce a choreography description language for Web-services. This group initially made little progress, so when the ADAPT interactions constraints language specification was required, at month 12 of the project, we had to produce a separate language. It is regrettable that we weren't able to participate in the standards effort more actively.

To allow runtime support for quality of service, performance properties were provided. When combined with the sensor system, this allowed composite services to monitor their component basic service, and alter their behavior if required, for example, making use of lightly loaded services.

1.2.3 Composability goal

The projects' achievements in the area of composability relate to two areas: (i) the design time use of service description (to capture information required to construct composite application) and (ii) the use of JOpera to allow the flexible composition of services (using the workflow systems capabilities to change the nature of the process and its support of integrating services at runtime).

1.2.4 Configuration goal

The ADAPT project did not, in the end, concentrate any effort on configurable software, that is to say, software whose behavior can be changed radically by changes to the configuration rather than changes to the code.

1.2.5 Adaptation goal

The two areas of adaptation that were addressed during the ADAPT project were: runtime adaptation to failure provided by the Basic Service Platform, and design time adaptation to service interfaces provided by the composite service execution engine.

The adaptation to failure supported by the Basic Services Platform provides failover between replicas and ensures that the session state maintained by the replicas are constant across the set of replicas. This is a powerful consistency property that many other systems do not possess. Converting a non-replicated service into a replicated service is a comparatively simple task; all that is required is a simple change to the service's deployment descriptor. The changes required for JBoss to support replicated services are more complicated, involving deployment and configuration of additional software and the use of a spread daemon. However, given that the changes need only be done once per machine then the task is not too much of an overhead.

The JOpera process manager provides support adapting the small differences in the syntactic and semantics between implementation of services [3][4]. This means that the ADAPT platform can, to a certain extent, allow inter-working of services that were not originally designed to work together. Of course if syntactic or semantic differences are too great then this would not be possible. The limits of what is possible with this approach have not been explored by the ADAPT project.

1.2.6 Process definition goal

One of the goals of the ADAPT project was to be able to flexibly specify processes so that the composite services could be easily constructed. When the project started there were numerous process definition standards, specifications seeking to be standards and non-standard specifications being promoted by industry. So initially it was not clear which of these process definition specifications, if any, would be suitable for the project. As a result, given the requirement to have a process definition language defined for a 12 month deliverable, the partners were forced to produce the ADAPT composition language, based on their experience in the field.

After a while BPDL4WS (which was later renamed WS-BPDL) became the leading standard in the area of defining processes between web services, but it was not initially clear what the legal status of the specification was or how stable the specification would be, as it was still being changed through the standards process. This standards process remains a protracted one, and it is still not clear if WS-BPEL will be adequate for the wide range of needs that it could be used for. Certainly, in its present from, it has limitations when it comes to using it for building composite services, of the type the ADAPT composition language was designed for. This is because the underling process model of WS-BPEL is the summation of a number of earlier specifications, which makes it hard to formally reason about the behavior of the resulting composite service.

It was decided to use JOpera as the composite service execution engine, this does not use the ADAPT composition language, but it does share a very similar underlying model.

1.2.7 Replication goal

The project concentrated its efforts on the replication of basic services, not the replication of composite services. This is not a real weakness of the ADAPT platform, because of the nature of composite services is to be long duration and containing long periods of inactivity. This means that the composite service engine’s ability to restart processes from the point that they had reached is therefore sufficient.

The replication of basic services was based around passive replication, so providing increased overall throughput but slower recovery failure than would be achieved using an active replication strategy. Given the application types and environments which the ADAPT platform is intended for, high load with rare failures, this seems a sensible approach for the ADAPT project.

1.2.8 Security goal

The ADAPT project did not, in the end, concentrate any effort on security.

1.2.9 Transaction models goal

The ADAPT project did not, in the end, concentrate on creating new transactional models, but rather put its efforts into building an advanced transaction engine and participating in standards bodies which are developing industry backed standards such as WS-CAF. The absence of a single standard for transaction coordination and a non-proprietary royalty free standard for the industry continues to be a roadblock to progress in the area. ADAPT has participated strongly in this standards arena putting great efforts behind the Web Services Composite Application Framework (WS-CAF) specification. Making progress has been difficult due to the set of competing specifications published by IBM/Microsoft/BEA on WS-C/T. However, during the last few months of the project, ADAPT, through Arjuna, has managed to continue to exert influence on the future roadmap for WS-T and achieved a significant breakthrough on 16 August 2005 when IBM and Microsoft invited Arjuna to be an author to their own specifications - Web Services Atomic Transaction (WS-AtomicTransaction) – This move brings the merger of the specifications much closer and will ensure that there will be a single standard for transaction coordination and that the standard will be non-proprietary and royalty free.

2 References

[1] "Demonstrator Specification", ADAPT project (IST-2001-37126), Deliverable D17, March 2004.

[2] "Demonstrator", ADAPT project (IST-2001-37126), Deliverable D20, September 2005.

[3] "Design and Evaluation of an Autonomic Workflow Engine", T. Heinis, C. Pautasso and G. Alonso, Proceedings of the 2nd International Conference on Autonomic Computing (ICAC-05), Seattle, Washington, June 2005.

[4] "Autonomic Execution of Service Compositions", C. Pautasso, T. Heinis and G. Alonso , Proceedings of the 3rd International Conference on Web Services (ICWS 2005), Orlando, Florida, July 2005.

Appendixes

The following documents, published by members of the ADAPT project, give further details of the evaluation of specific parts of the overall ADAPT platform. These documents are:

Appendix 1

- MIDDLE-R: Consistent Database Replication at the Middleware Level, *Marta Patio-Martiñez[1], Ricardo Jiménez-Peris[1], Bettina Kemme[2] and Gustavo Alonso[3]*, ACM Transactions on Computer Systems (TOCS). In Press.

Appendix 2

- Middleware based Data Replication providing Snapshot Isolation, Yi Lin[2], Bettina Kemme[2], Marta Patiño-Martiñez[1] and Ricardo Jiménez-Peris[1], ACM SIGMOD Int. Conf. on Management of Data. Baltimore, Maryland, USA. Jun. 2005.

Appendix 3

- A Framework for Prototyping J2EE Replication Algorithms, *Özalp Babaoğlu1[4], Alberto Bartoli[5], Vance Maverick[4], Simon Patarin[4], Jakša Vučković[4] and Huaigu Wu[2]*, International Conference on Distributed Objects and Applications (DOA) 2004, pages 1413-1426.

Appendix 4

- Consistent Data Replication: Is it feasible in WANs? *Yi Lin[2], Bettina Kemme[2], Marta Patñio-Martiñez[1] and Ricardo Jiménez-Peris[1]*, Europar Conf., Lisbon (Portugal), August 2005.

Appendix 5

- Adaptive Middleware for Data Replication, J. M. Milan-Franco[1], R. Jiménez-Peris[1], M. Patiño-Martiñez[1], and B. Kemme[2], ACM/IFIP/USENIX Middleware Conference, Toronto (Canada). Oct. 2004.

Appendix 6

- Eager Replication for Stateful J2EE Servers, *Huaigu Wu[2], Bettina Kemme[2], and Vance Maverick[4]*, Int. Symposium on Distributed Objects and Applications (DOA) 2004.

Appendix 7

- Fault-tolerance for Stateful Application Servers in the Presence of Advanced Transactions Patterns, *Huaigu Wu[2] and Bettina Kemme[2]*, Proc. Int. Symposium on Reliable Distributed Systems (SRDS) 2005.

Appendix 8

- JOpera: a Toolkit for Efficient Visual Composition of Web Services, *C. Pautasso[3] and G. Alonso[3]*, International Journal of Electronic Commerce (IJEC), 9(2):107-141, Winter 2004/2005

Appendix 9

- Design and Evaluation of an Autonomic Workflow Engine, *T. Heinis*[3]*, C. Pautasso*[3] *and G. Alonso*[3], In: Proceedings of the 2nd International Conference on Autonomic Computing (ICAC-05), Seattle, Washington, June 2005.

Appendix 10

- Autonomic Execution of Service Compositions, *C. Pautasso*[3]*, T. Heinis*[3]*, and G. Alonso*[3], In: Proceedings of the 3rd International Conference on Web Services (ICWS 2005), Orlando, Florida, July 2005.

Appendix 11

- Highly Available Long Running Transactions and Activities for J2EE Applications, *Francisco Perez, Jaksa Vuckovic, Marta Patiño-Martinez*[1]*, and Ricardo Jimenez-Peris*[1]. Technical Report. Universidad Politecnica de Madrid. December 2004.

(1) Facultad de Informática, Universidad Politécnica de Madrid (UPM), Madrid, Spain
(2) School of Computer Science, McGill University, Montreal, Canada
(3) Department of Computer Science, Swiss Federal Institute of Technology (ETHZ) Zürich, Switzerland
(4) Università di Bologna, Bologna, Italy
(5) Università degli Studi di Trieste, Trieste, Italy