The three papers in this session each address different aspects of the provision of rich environments for persistent application development. The first, which was presented by Erik Vogues, looks at the provision of high level tools for spatiotemporal querying. The second, presented by Nicolas Richer, addresses support for distributed persistent applications. The third, presented by Alan Kaplan, looks at heterolingual persistent programming.

In his talk Vogues motivated the need for spatiotemporal querying with the example of a GIS used for fisheries research. He explained that spatiotemporal query classes written in Java, layered over PJama (an orthogonally persistent Java), provided a suitable infrastructure for the construction of a spatiotemporal GIS. He argued that this approach was better than building the GIS directly over either PJama or a relational GIS—the encapsulation provided by the spatiotemporal class greatly aided construction of the GIS, and the abstraction over persistence provided by PJama in turn substantially simplified construction of the spatiotemporal class.

Olivier Gruber asked for more details about the performance analysis, in particular, what size the systems were. Vogues explained that they had not yet performed rigorous analysis but that the largest system was about 1GB. Ole Anfindsen asked about the role of PJama in the system, and what the differences might be from the standpoint of the application developer if the underlying system had been a relational database with support for Java. Vogues responded by saying that he thought that relational query languages and the object-oriented design approach were somewhat at odds. Anfindsen then asked whether a query example presented in the talk could have been implemented using a relational query. Vogues said he thought it could have been, but went on to say that some more complex queries might have been hard to implement using a relational query language.

In the second talk, Richer described PerDIS as a new technology for sharing over the internet—a Persistent Distributed Store. PerDIS is targeted at applications such as distributed GIS, and was motivated through an example based on an internationally-distributed building project involving architecture, engineering, etc. The focus was on constructing a system that allowed sharing in such an environment. Central to the approach was the use of ‘domains’ as a means of limiting the scope of coherency operations. Intra-domain coherency is maintained transactionally (either ‘pessimistically’ or ‘optimistically’), but inter-domain coherency is manually mediated using a simple check-in, check-out model. Caching is performed at various grains adaptively. The PerDIS model is one of a network-wide persistent shared memory, and so is amenable to both distributed application construction, and the hosting of legacy applications. The persistent shared memory space is garbage collected.
Richard Jones asked whether the inability of Larchant (the garbage collector used by PerDIS) to collect cycles had proved to be a problem. Richer said that it was indeed a problem and that in fact he was presenting a paper later in the proceedings that addressed that issue. Ole Anfindsen asked whether failures that arise due to optimistic transaction commit conflicts or the need for manual intervention during check-in amounted to a serious problem for PerDIS. Richer explained that the problem was not negligible, and pointed out that the semantics of reconciliation in the check-out model necessarily involved manual intervention.

During the panel session Ole Anfindsen asked whether the lack of support for distributed transactions was a serious restriction. Richer replied that for the applications they were targeting, most concurrency issues occurred intra-site (for example between cooperating engineers), rather than inter-site (an architect might only require read only access to the engineers’ work). He said that although inter-site transactions would enrich PerDIS, such an addition would be a major undertaking.

In his presentation, Alan Kaplan explained pure polylingual persistence by contrasting it with homogeneous (mono-lingual) persistent systems, segregated heterolingual systems, polylingual persistent systems where object closures must be mono-lingual (such as that provided by PolySPIN) and pure polylingual systems, where object closures may be heterolingual. He then presented \( P^3 \), a general framework for pure polylingual persistence. \( P^3 \) assumes reachability-based orthogonal persistence, where each language is responsible for the persistence of objects it instantiates. He explained how heterolingual objects (objects that contain pointers to objects from different languages) are implemented through the use of proxies. In support of the case that pure polylingual persistence is viable, performance results were presented for a Java/C++ polylingual implementation of OO7.

Tony Hosking asked how \( P^3 \) might be applied to languages which are significantly different. Kaplan said that while the application of \( P^3 \) to languages with similar type systems such as C++, Java, Ada95 or Eiffel was fairly clear, he was not sure how it would be applied to languages with very different type systems, such as ML and Java. Graham Kirby then asked whether consideration had been given to mixing languages at the class rather than instance level. The response was yes, but there were a number of interesting issues that needed to be addressed, particularly with respect to inter-lingual inheritance. Al Dearle then asked where in the system the proxy mechanisms were manifest. Kaplan explained that the proxy mechanisms did have to be written into the code, but that their system tried to make the insertion of this code transparent to the application programmer. Ewa Bem asked whether a language-neutral storage format had been considered. Kaplan said that they felt that any universal representation is unlikely to be optimal for any language and that the cost of transformation would have to be weighed against the overhead of proxies used for inter-lingual accesses.

During the panel session that followed the talks Malcolm Atkinson asked all of the speakers how they might measure the utility of what each of their systems added to persistent environments (spatiotemporal querying, distribution, polylingual capabilities). All of the speakers agreed that this was an important issue, but one that is particularly hard to quantify. In particular, it is hard to calibrate the tradeoff between usability and programmability enhancements (development) and any performance impact (runtime).