

CS6201 Software Reuse

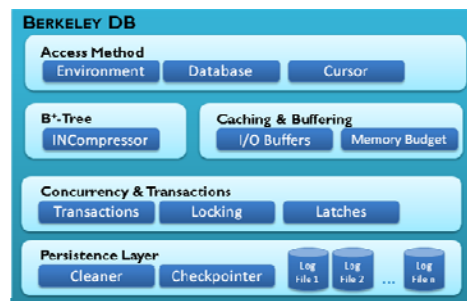
CS6201 Lecture Notes: Berkeley DB case study

Berkeley DB

- Berkeley DB: database engine, 86KLoC

Five DB subsystems:

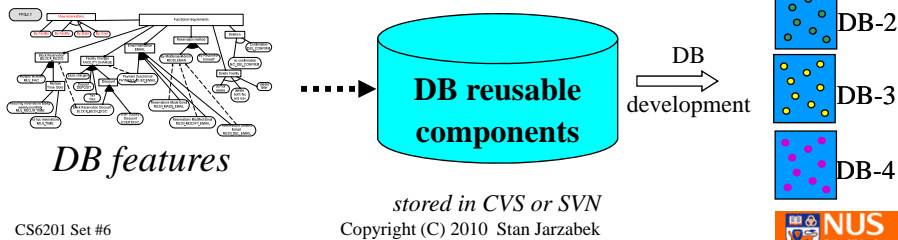
Base components of DB



- 38 DB variant features: AtomicTransaction, EnvironmentLock, FSync, LatchesLeak, CheckingStatistics, ...
- Each DB product variant may contain some combination of 38 features – many DB product variants

Towards software Product Line (SPL) systematic reuse

- Understand DB similarities and differences DB
 - Top-down, business-oriented analysis of user needs
 - Bottom analysis of existing DB products
- Standardize DB architecture and components
- Design parameterized, reusable DB components
 - Apply variation mechanisms to facilitate reuse



CS6201 Set #6

stored in CVS or SVN
Copyright (C) 2010 Stan Jarzabek



3

Examples of DB features

Feature	Description
AtomicTransaction	Part of the transaction system that is responsible for atomicity
EnvironmentLock	Presents two instances on the same database directory
FSync	File synchronization for writing log files
Latches	Fine grained thread synchronization
LeakChecking	Debug checks for leaking transactions
Statistics	Collects runtime statistics like buffer hit ratio throughout the system

CS6201 Set #6

Copyright (C) 2010 Stan Jarzabek



4

Feature impact on DB components

Feature	# DB base components affected	# variation points
MemoryBudget	32	190
Evictor	12	28
Checksum	10	28
Statistics	10	34
CheckPointer	5	34
CpByteConfig	4	6
CpTimeConfig	4	7

CS6201 Set #6

Copyright (C) 2010 Stan Jarzabek



5

Feature interactions

Feature	Interacting feature	# variation points
CheckPointer	Statistics	22
MemoryBudget	Evictor	5
MemoryBudget	CriticalEviction	1
SyncIO	IO	4
EvictorDaemon	Evictor	3

CS6201 Set #6

Copyright (C) 2010 Stan Jarzabek



6

Feature impact on FileProcessor

```
public class FileProcessor {
    ...
    private boolean processFile(...) {
        ...
        LookAheadCache lookAheadCache = new LookAheadCache();
        ...
        lookAheadCache.add(getFileOffset(...));
        ...
        if (lookAheadCache.isFull()) {
            processLN(., lookAheadCache, ..);
        }
        ...
    }
    private processLN(..., LookAheadCache lookAheadCache, ...) {
        int offset = lookAheadCache.getOffset();
        ...
    }
}
```

CS6201 Set #6

Copyright (C) 2010 Stan Jarzabek



7

Feature management in SPL

- **Feature** may mean any product characteristic
- One **feature** may affect many product components

Features interactions:

- **Functionally interdependent features:**
 - If I select one feature I must also select some other features
- One feature may **affect implementation** of other features

CS6201 Set #6

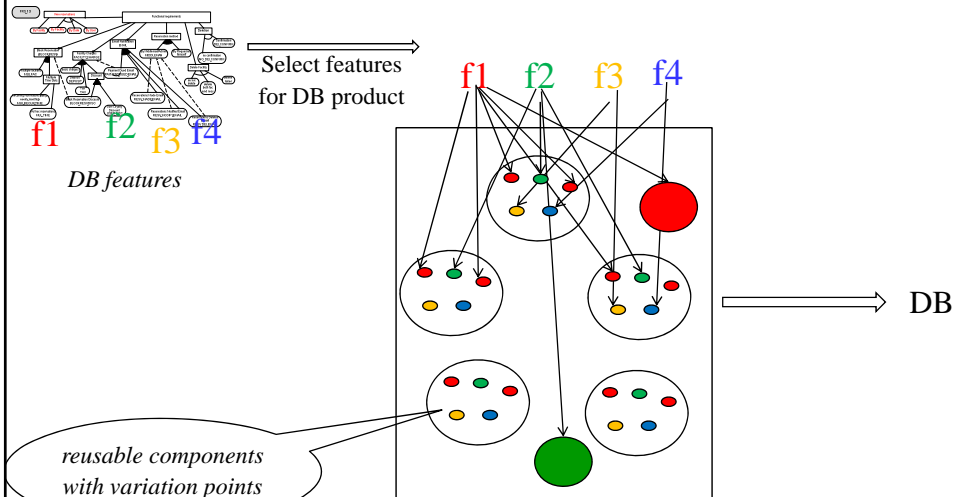
Copyright (C) 2010 Stan Jarzabek



8

Reuse challenge:

Tracing the impact of features on components



CS6201 Set #6

Copyright (C) 2010 Stan Jarzabek



9

Component reuse is not enough

Variation mechanisms help manage impact of features on components

- Runtime variation mechanisms
 - Design patterns, reflection, conditionals
- Construction time variation mechanisms
 - Parameterization: generics, templates, ...
 - Configuration parameters
 - Preprocessing (#ifdef), commenting out feature code
 - Build tools (make, Ant), Wizards
 - Aspect-Oriented Programming (AOP)
 - check Kästner, C., Apel, S. and Batory, D. "A Case Study Implementing Features Using AspectJ," Proc. Int. Software Product Line Conference, SPLC'07, Kyoto, 2007, pp.223-232

CS6201 Set #6

Copyright (C) 2010 Stan Jarzabek



10

Berkeley DB SPL in XCpp

- XCpp is a subset of XVCL
 - XML-based Variant Configuration Language
 - A construction time variation mechanism for SPL
 - All-in-one solution to managing features in SPL
 - Used in sync with conventional programming technologies:
 - Java/XVCL, ASP/XVCL, PHP/XVCL, J2EE/XVCL, .NET/XVCL
- Public domain, available at sourceforge

<http://xvcl.comp.nus.edu.sg>

```
SPC // specifies feature selection for DBNEW
<set a = "A" />
<set d = "D" />
<set b = ""/> <set c = ""/>
<adapt DBSchema/>
<adapt FeeUser/>
```

Suppose A, B, C, D are all DB features
and we selected A and D for DB^{NEW}

```
FileManager
<set v = @a />
<select v > // feature A affects FileManager here
<option A> if feature A is selected
<otherwise> if A is not selected
</select>
...some code for FileManager
<set v = ..>
<select v > // another variation point in FileManager
```

```
Evictor
public class Evictor{
...some code for Evictor
<set v = @a @c @d />
<select v > // feature s A, C and D affect Evictor here
<option A> code for feature A
<option A D> code for feature interaction A and D
<option A C D> code for feature interaction A, C and D
<otherwise>
</select>
... some code for Evictor
<set v = .. >
<select v > // another variation point in Evictor
```

```

SPC // here we set parameters for features required in DBNEW
<set all_features = "IO, EvcitorDaemon, LookAheadCache, DiskFullHandler,
Evictor, MemoryBudget ..."/>

<while all_features> <set @all_features = "" /> </while>

// Features selected for DBNEW
<set selected_features = "IO, LookAheadCache, DiskFullHandler, Evictor"/>

<while selected_features> <set @selected_features = @selected_features/> </while>

<adapt FileManager />
<adapt Evictor/>

```

```

FileManager
public class FileManager .. {
...
<set v = @LookAheadCache />
<select v>
<option LookAheadCache >
    code for LookAheadCache
</option>

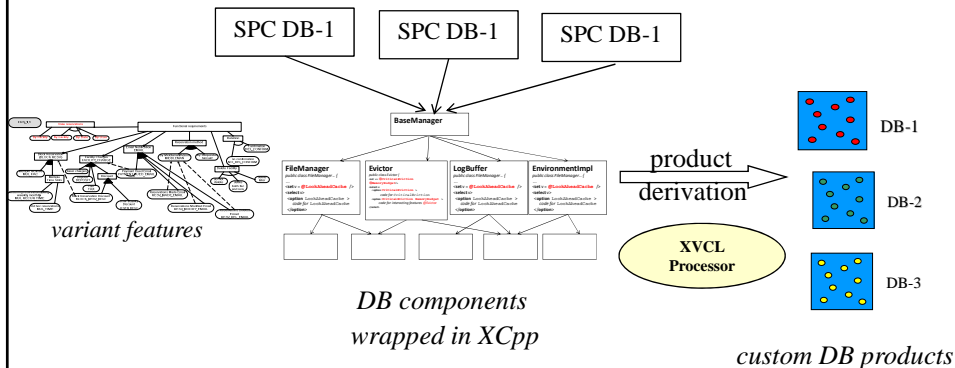
```

```

Evictor
public class Evictor {
<set v = @CriticalEviction @MemoryBudget/>
<select v>
<option CriticalEviction >
    code for CriticalEviction
<option CriticalEviction MemoryBudget >
    code for interacting features @Evictor
</select>

```

Generation custom DB products



Problem still remains:

- How to find all variation points in many components relevant to a given feature?

Feature queries

- FQL: Feature Query Language
- A tool locates and shows all variation points relevant to a given feature
- Show all variation points where feature “f” affects components

```
Declare option o
Select o
  where o.feature="f"
```

- Show all variation points where feature “f” interacts with other features

```
Declare option o
Select o
  where o.feature="*f*"
```

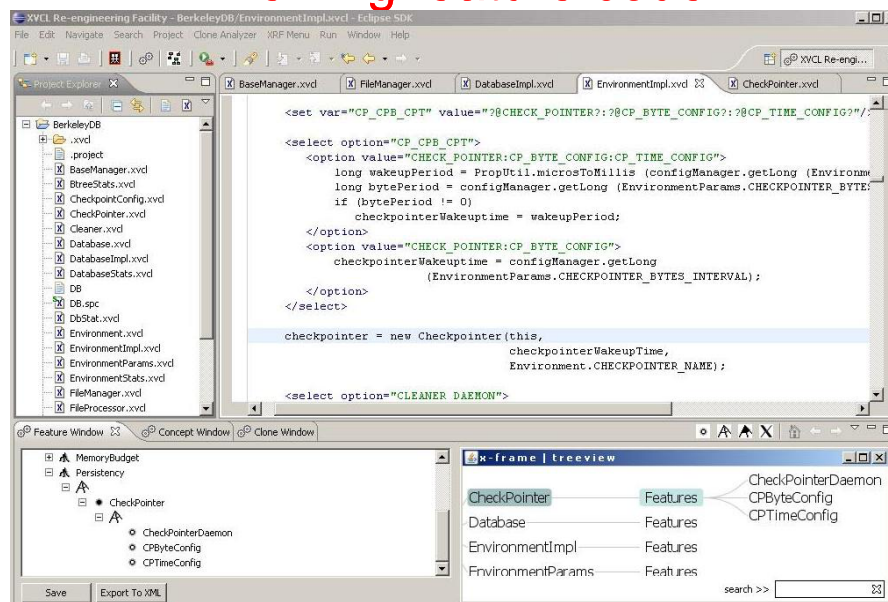

Filtering feature views with queries

- Where do features Statistics and MemoryBudget interact with each other?

```
Declare option o
Select o
where o.feature = "*STATISTICS*" and
o.feature = "*MEM_BUDGET*"
```

```
<set stat_membudget="?"@STATISTICS?_?@MEM_BUDGET?"/>
<select option="stat_membudget">
  <option value="STATISTICS_MEM_BUDGET">
    long getFreeMemory() {
      maxMemory - usedMemory;
    }
  </option>
</select>
```

Viewing feature code



The screenshot shows the Eclipse IDE interface. The main editor displays the feature code for the 'CheckPointer' feature, which includes configuration options for wakeup period and interval, and a call to the 'new Checkpointer' constructor. The 'Project Explorer' on the left shows the project structure for 'BerkeleyDB'. The 'Feature Window' at the bottom left shows a tree view of features, with 'CheckPointer' expanded to show its sub-features: 'CheckPointerDaemon', 'CPByteConfig', and 'CPTimeConfig'. The 'x-frame | treeview' window at the bottom right shows a dependency graph where 'CheckPointer' is connected to 'Features', which in turn is connected to 'CheckPointerDaemon', 'CPByteConfig', and 'CPTimeConfig'.

Summary of approach

- Embed features in reusable components
- A mechanism to *compose* required features into the product
- Mark each variation point with names of interacting features
- Formally inter-link all variation points affected by a given feature
- Query-based visualization of features and their interactions

Evaluation: problems solved

- Legality of feature selection
 - Validation done prior to feature processing (Zhang, H)
- Automation of product derivation
 - feature composition into base done by XVCL Processor
- Feature comprehension
 - each variation point marked with names of interacting features
 - inter-linking all variation points affected by a given feature
 - query-based visualization of features and their interactions

Remaining problems

- Solution gets complicated as the size of product increases, and the number of features and feature dependencies grows
 - True, we can find feature code – but how to understand, maintain and reuse features if their code spreads though many variation points, in many base components?
- Assumption of “base components” is limiting
 - we can’t contain the impact of features at the implementation level only – use design!
- Direction for future work:
 - Reduce the number of variation points
 - Relax the assumption of a “base components”
 - Represent products in generic form (full XVCL)

Q & A



End of Berkely DB case study