マルチパラダイムデザインと C++における実装

Multi-Paradigm Design and Implementation in C++

James O. Coplien

Distinguished Member of Technical Staff
Silicon Prairie Research Department, Bell Laboratories
Naperville, Illinois, USA

Visiting Professor, University of Manchester Institute of Science and Technology
Manchester, United Kingdom

パラダイムとは?

- モノを組織化する1つの方法
- 抽象化により，組織化する
- 抽象化では，共通する何かに焦点をあてる
- バリエーションは個別に扱う

C++ != OO

- Stroustrup は，C++ がオブジェクト指向プログラミング言語であるとは言っていない
- 実践的C++プログラマの大半が，OO手法を使っている
- C++ は，OO言語というだけではない
- C++ には，その特性を活かすことのできるような手法が必要である
- マルチパラダイムデザイン: (特定のプログラミング言語の宇宙の中で) “ユニバーサル”なパラダイム
- C++以外の言語にも，マルチパラダイムデザインを適用することができる

マルチパラダイムデザイン

- ドメイン分析
- 共通性分析と可変性分析
- 共通性と可変性:
  - アプリケーションドメイン（「問題」）における共通性，可変性
  - ソリューション(プログラミング言語)における共通性，可変性
- アプリケーション分析をソリューション分析に
  マッピングする
- 正規化のために，パターンとイディオムを識別する
- 汎用化のために，OOAを識別する

パートI　：　ドメイン分析

- ドメイン分析は，分析と等しい拡がりを持つ
- マルチパラダイムデザイン：豊かなドメイン
  - アプリケーションサブドメイン分析
  - ソリューションドメイン分析
- ソフトウェアファミリでシステムを組織化する
- ファミリは1個のドメインを形成する

ドメインの特性

- 直観とビジネス経験に従う
- 目前のアプリケーションではなく，ビジネスの境界に合わせて分析する
  - よりメンテナンス性の高いシステムを導く
  - 再利用を支援する可能性
- ドメインとモジュール分割
  - 最良のドメインは完全に分解されたモジュール
  - ドメインはオーバーラップする

例　テキストエディタのためのドメイン

- コマンド：
  - マルチ編集言語になる
- バッファ：
  - さまざまなメモリ管理スキームを備える
- ファイル：
  - さまざまなキャラクタセットとフォーマットを支援する
- キーボード：
  - ファンクションキー/矢印キーの有無
- スクリーン：
  - さまざまなテクノロジー

パートII　：　共通性分析

- 抽象の本質
- 共通性はファミリ構成員からファミリを定義する
- 分割基準は抽象から生じるのであって，逆ではない
- 共通性には多数の軸が存在する:
  - 振る舞い
  - データ構造
  - 名前
  - コード構造
- 共通性カテゴリとして，これらの軸を利用できる

共通性の例：　テキストバッファ

- 構造:
  - ライン数
  - カレントライン
  - 名前
- 振る舞い:
  - ファイルもしくはストリームからデータを取り込む
  - ファイルもしくはストリームにデータを書き込む
  - 指定されたラインを取り出す
  - 内部メモリ管理

Part III　：　可変性分析

- 可変性とは，共通性の不在
- 可変性は，ファミリ構成員を識別する
- 可変性は，不可抗力
- 可変性 != 詳細!
- 可変性をパラメータ化する
- 可変性の軸は，共通性と同じものを利用する(構造，振る舞い，型，名前)：共通性カテゴリ
- 可変性分析は，共通性分析と並行して進捗する

可変性の例: テキストバッファ

- ワーキングセットアルゴリズム
  - LFU，LRU，ページ，ファイル
  - 注記: 振る舞いは共通している!
- キャラクタセット
  （ASCII, EBCDIC, FIELDATA, char, wchar_t
  ）
- バージョン化されている/されていない

可変パラメータ

- 可変性をパラメータ化したい
- 可変パラメータを定義する
  - ワーキングセットアルゴリズム
  - キャラクタセット
  - バージョニング
- 可変パラメータは，ファミリの「遺伝コード」
- 各パラメータを「値」で置き換えることによって，ファミリ構成員を生成することができる.

正の可変性と負の可変性

- 正の可変性はその基底をなす共通性を損わない
  - Public 継承は基底クラスの振る舞いを保持する
  - テンプレートのインスタンスは共通構造を保持する
- 負の可変性は共通性を侵害する
- ソリューションドメインの例:
  - #ifndef
  - オーバーライド関数のデフォルト引数
  - テンプレートの特殊化
  - キャンセルを伴う継承（private 継承）

アプリケーション分析とソリューション分析

- 問題ドメインは，共通性と可変性を提示する
- ソリューションドメインもまた，すべてのレベルにわたって共通性と可変性を提示する
- プログラミング言語は，共通性と可変性の対を表現する
  - 継承: 共通の振る舞い，異なるデータ構造とアルゴリズム
  - テンプレートT: 共通の構造，タイプごとに異なる振る舞い
  - オーバーローディング(多重定義): 共通の名前とセマンティクス；異なるアルゴリズムとパラメータ
  - . . . .
- ソリューションドメイン分析は独特のアクティビティ

ソリューションドメイン分析

- アプリケーションの共通性カテゴリと可変性カテゴリの対がパラダイムを構成する
  - 抽象化により，ソフトウェアを組織化する
  - 抽象化では，共通するものに焦点をあてる
- オブジェクトパラダイム
  : 共通する構造と振る舞い; 可変の構造とアルゴリズム
- オーバーロードされた手続きはファミリを形成する
  : 共通する名前; 可変のアルゴリズム
- テンプレートのインスタンスはファミリを形成する
  : 共通する構造; 異なる型　など
- 注記: Tim Budd のパラダイムとは異なる

変換分析テーブル

Commonality	Variability	Binding	Instantation	C++ Feature
Function Name and Semantics	Anything other than algorithm structure	Source	N/a	Template
	Fine algorithm	Compile	N/a	#ifdef
	Fine or gross algorithm	Compile	N/a	Overloading
Data Structure	Value of State	Run Time	Yes	Struct, simple types
	A small set of values	Run time	Yes	Enum
	Types, values and state	Source	Yes	Template
Related Operations and Some Structure	Value of State	Source	No	Module
	Value of State	Source	Yes	struct, class
	Data Structure and State	Compile	Optional	Inheritance
	Algorithm, Data Structure and State	Compile	Optional	Inheritance
		Run	Optional	Virtual Functions

マルチパラダイムデザイン: 変換分析

- アプリケーションをサブドメイン候補に分割する
- そのアプリケーションドメインの共通性と可変性を，テーブルを用いて表現する
- カタログ化されているプログラミング言語の共通性と可変性を用いて，それらを調節する.
- 変換分析の観点から分析構造を実装する

例: テキストバッファ

- 共通性: 振る舞い，データ構造の一部
- 可変性:
  - キャラクタセット(型)
  - ワーキングセット管理(アルゴリズム)
- ソリューション:
  - テンプレート (共通するデータ構造, 異なる型)
  - 仮想関数を持つ継承 (共通する振る舞い，異なる機能)

TextBuffer 変換分析

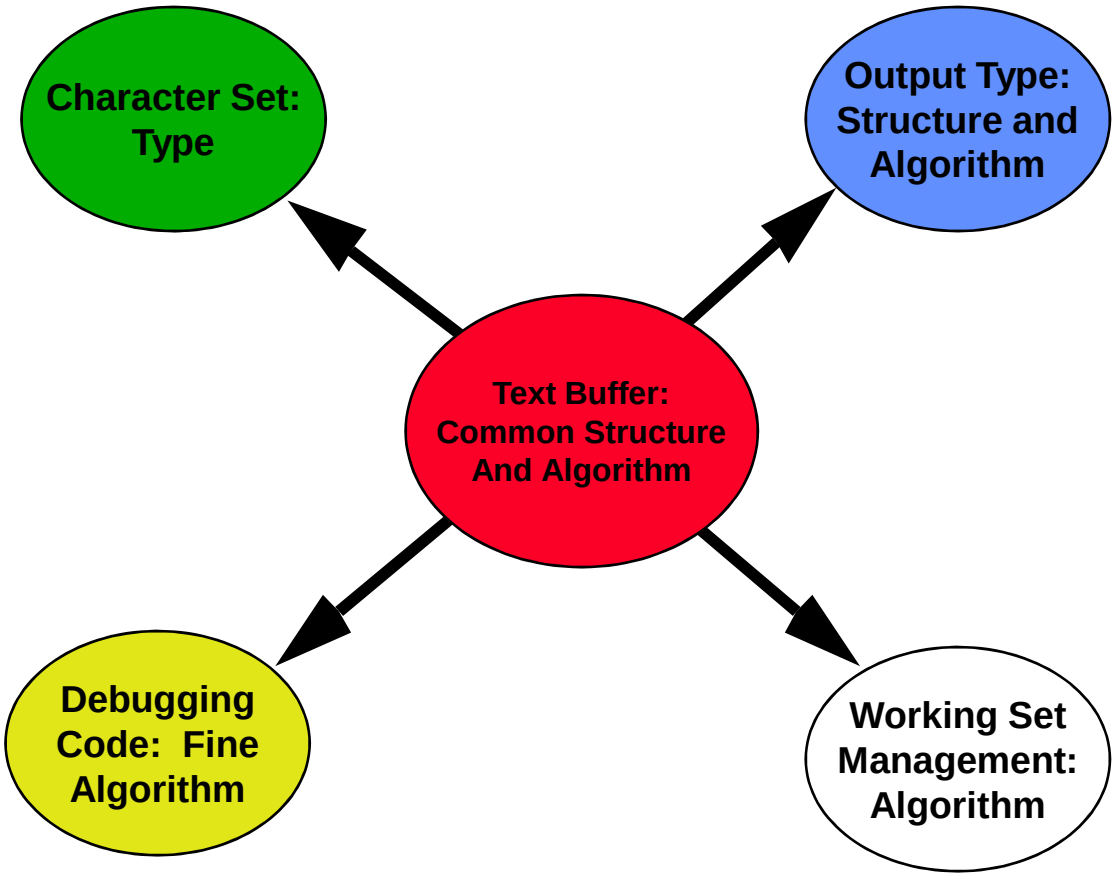
TextBuffer: 共通する構造とアルゴリズム

Parameters of Variability	Meaning	Domain	Binding	Default Technique
Output Type Structure, *Algorithm*	The formatting of text lines is sensi-tive to the output medium	Database, RCS, TTY, UNIX file	Run	UNIX File *Virtual Functions*
Character Set *Non-structural*	Different buffer types support different character sets	ASCII, EBCDIC, FIELDATA	Compile	ASCII *Templates*
Working Set Management *Algorithm*	Different applications need to cache dif-ferent amounts of memory	Whole file, whole page, LRU fixed	Compile	Whole file *Inheritance*
Debugging Code *Code Fragments*	Debug in-house only, but keep tests in source code	Debug, production	Compile	None *#ifdef*

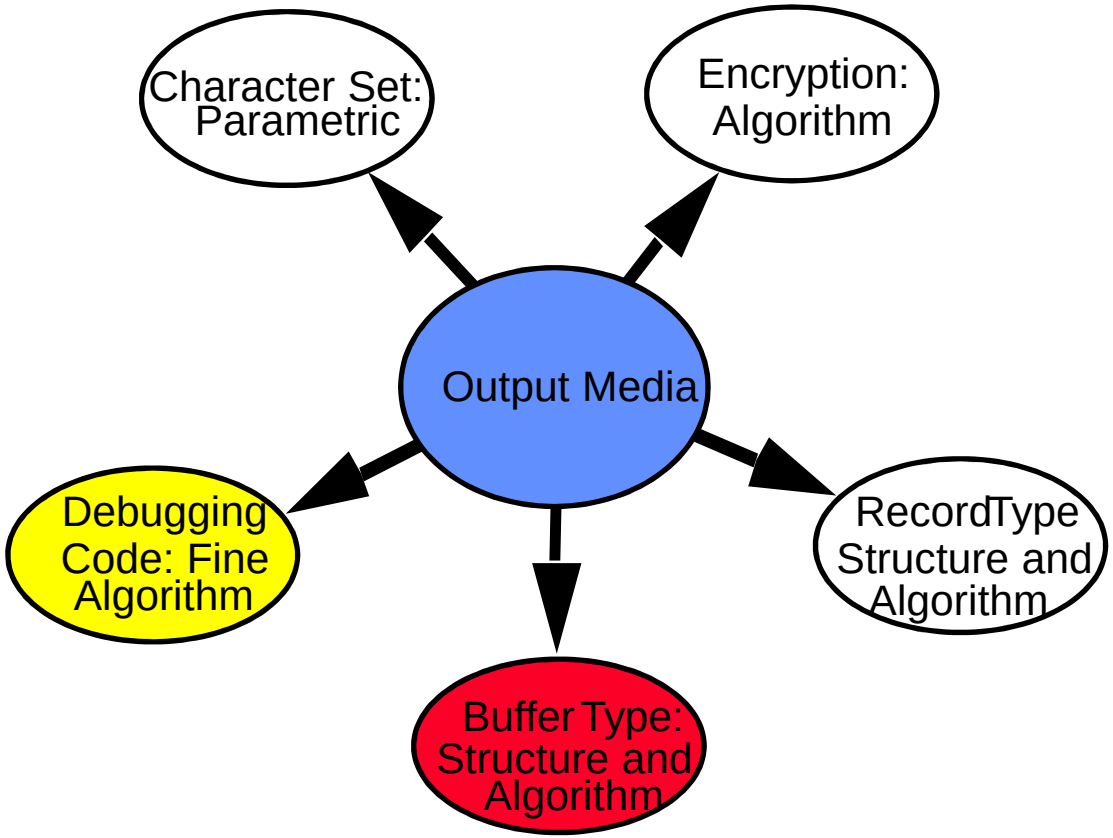
テキストバッファのソリューション

```
template <class CharSet> struct TextBuffer {
        virtual Line line(const LineNumber&) const;
        virtual void write(File&);
        . . . .
private:
        virtual void pageManagement(const LineNumber&);
};

class EmacsBuffer1: public TextBuffer<wchar_t> {
        void pageManagement(const LineNumber &);
        . . . .
};

class EmacsBufferJap: public TextBuffer<Katakana> {
        void pageManagement(const LineNumber &);
};
```

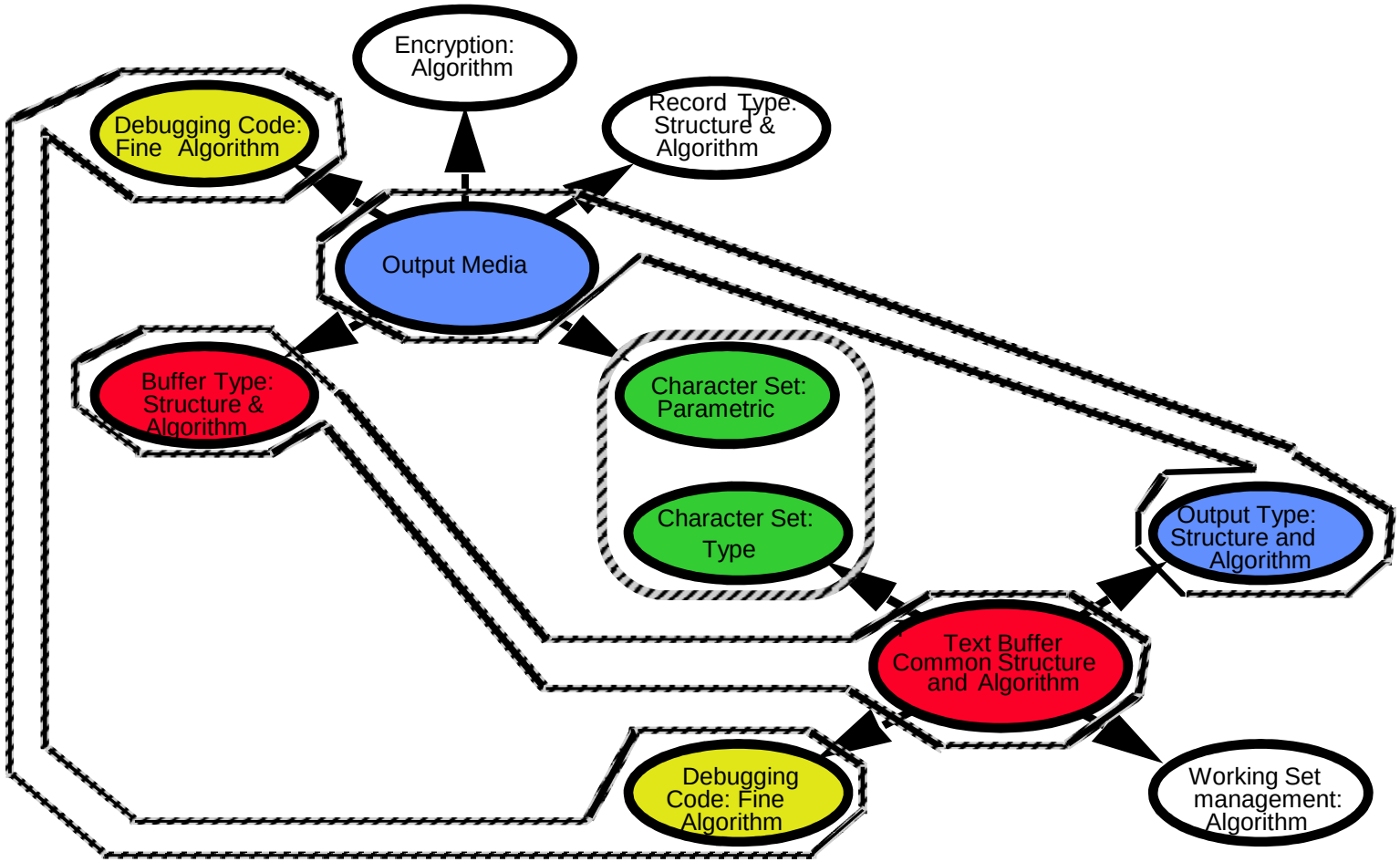
テキストバッファの依存性グラフ



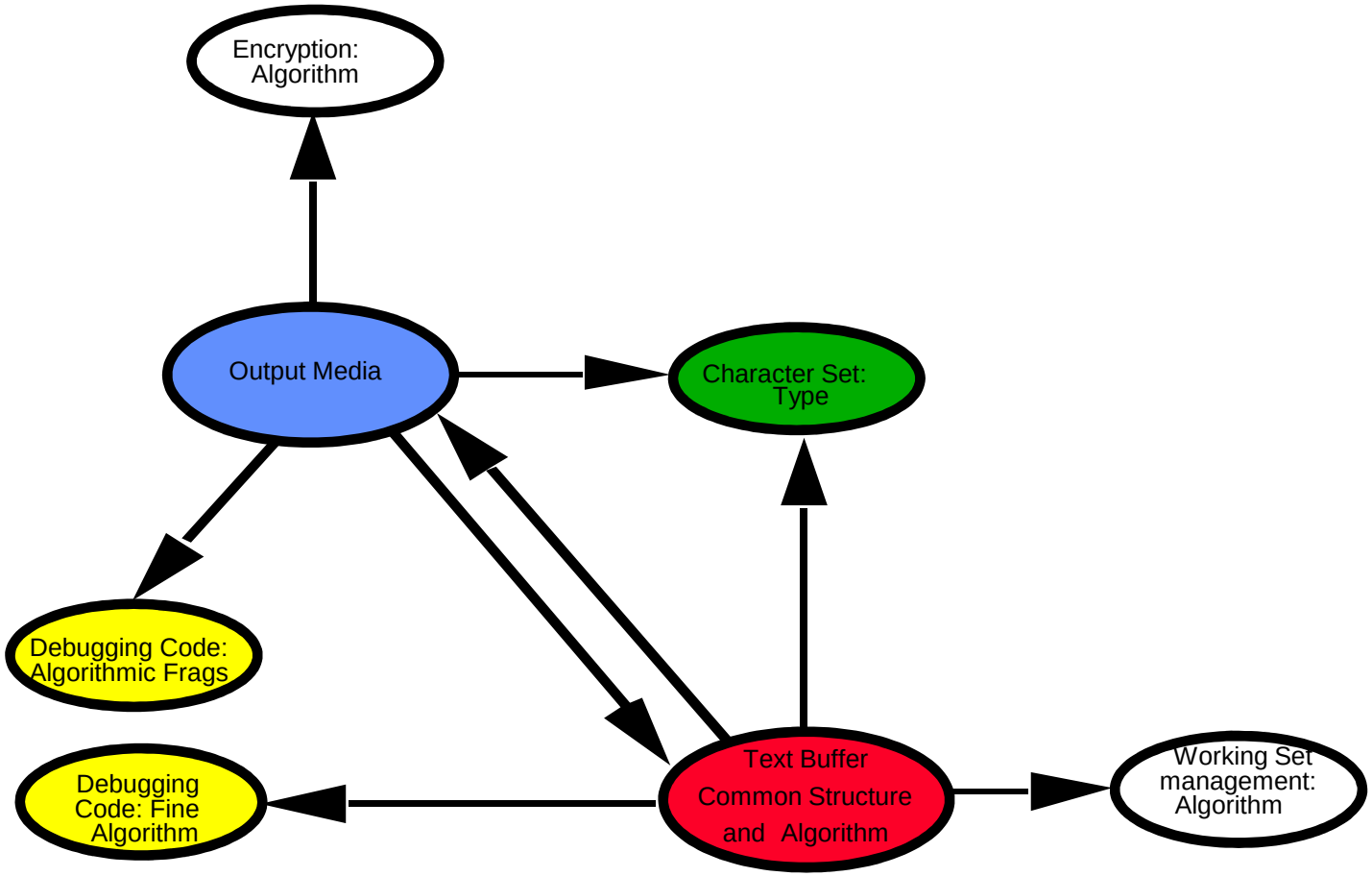
ファイルドメイン


Character Set: Parametric
Encryption: Algorithm
Output Media
Debugging Code: Fine Algorithm
RecordType Structure and Algorithm
Buffer Type: Structure and Algorithm

多重ドメインの循環依存性サイクル



設計を統一する


Encryption:
Algorithm
Output Media
Character Set:
Type
Debugging Code:
Algorithmic Frags
Debugging
Code: Fine
Algorithm
Text Buffer
Common Structure
and Algorithm
Working Set
management:
Algorithm

１つのソリューション

```
template <class TextBuffer, class CharSet>
class OutputMedium {
public:
  void write() {
       . . . .
       subClass->getBuffer(writeBuf);
  }
  OutputMedium(TextBuffer *sc): subClass(sc) { }
protected:
  TextBuffer *subClass;
  CharSet writeBuf[128];
};

template <class TextBuffer, class Crypt, class
  CharSet>
class UnixFile: public OutputMedium<TextBuffer,
  CharSet>,
  protected Crypt {
public:
  UnixFile(TextBuffer *sc):
       OutputMedium<TextBuffer, CharSet>(sc) { }
  void read() {
       . . . .
       Crypt::decrypt(buffer);
  }
};
```

TextBuffers

```
template <class CharSet>
class TextBuffer {
public:
  string getLine() {
       string retval;
       . . . .
       return retval;
  }
  void getBuffer(CharSet *) { . . . . }
  TextBuffer() { . . . . }
};

template <class Crypt, class CharSet>
class UnixFilePagedTextBuffer: public
  TextBuffer<CharSet>,
  protected
  UnixFile<UnixFilePagedTextBuffer<Crypt,CharSet>,
       Crypt, CharSet> {
public:
  UnixFilePagedTextBuffer(): TextBuffer<CharSet>(),
       UnixFile<UnixFilePagedTextBuffer<Crypt,CharSet>,
            Crypt, CharSet>(this) { . . . . }
  string getLine() { . . . . read(); . . . . }
};
```

暗号化; main

```
class RSA {
protected:
  void encrypt(string &);
  void decrypt(string &);
};


int main() {
  UnixFilePagedTextBuffer<RSA, wchar_t> buffer;
  string buf = buffer.getLine();
  . . . .
}
```

例: 状態遷移マシン

- AbstractFSM ドメイン: 状態遷移マシンの本質的な振る舞い
- UserFSM ドメイン: 個別のFSMに特化した構造．状態，遷移，アクションを備える
- GenericFSM ドメイン: すべてのFSM実装に共通する構造 (遷移テーブルなど)
- State, Stimulus: FSMで一般的に使用されるもの

AbstractFSM の可変性テーブル

共通性: 構造(Structure)と振る舞い(Behavior)

Parameters of Variation	Meaning	Domain	Binding	Default
UserFSM *Structure, algorithm*	All generic FSMs understand how to add transitions in any UserFSM	Any class having member functions accepting a Stimulus argument	Compile time	None *Templates*
State *Type*	How to represent the FSM state	Any discrete type	Compile time	None *Templates*
Stimulus *Type*	The type of the message that sequences the machine between states	Any discrete type	Compile time	None *Templates*

ImplementationFSM 可変性テーブル

共通性: 構造(Structure)と振る舞い(Behavior)

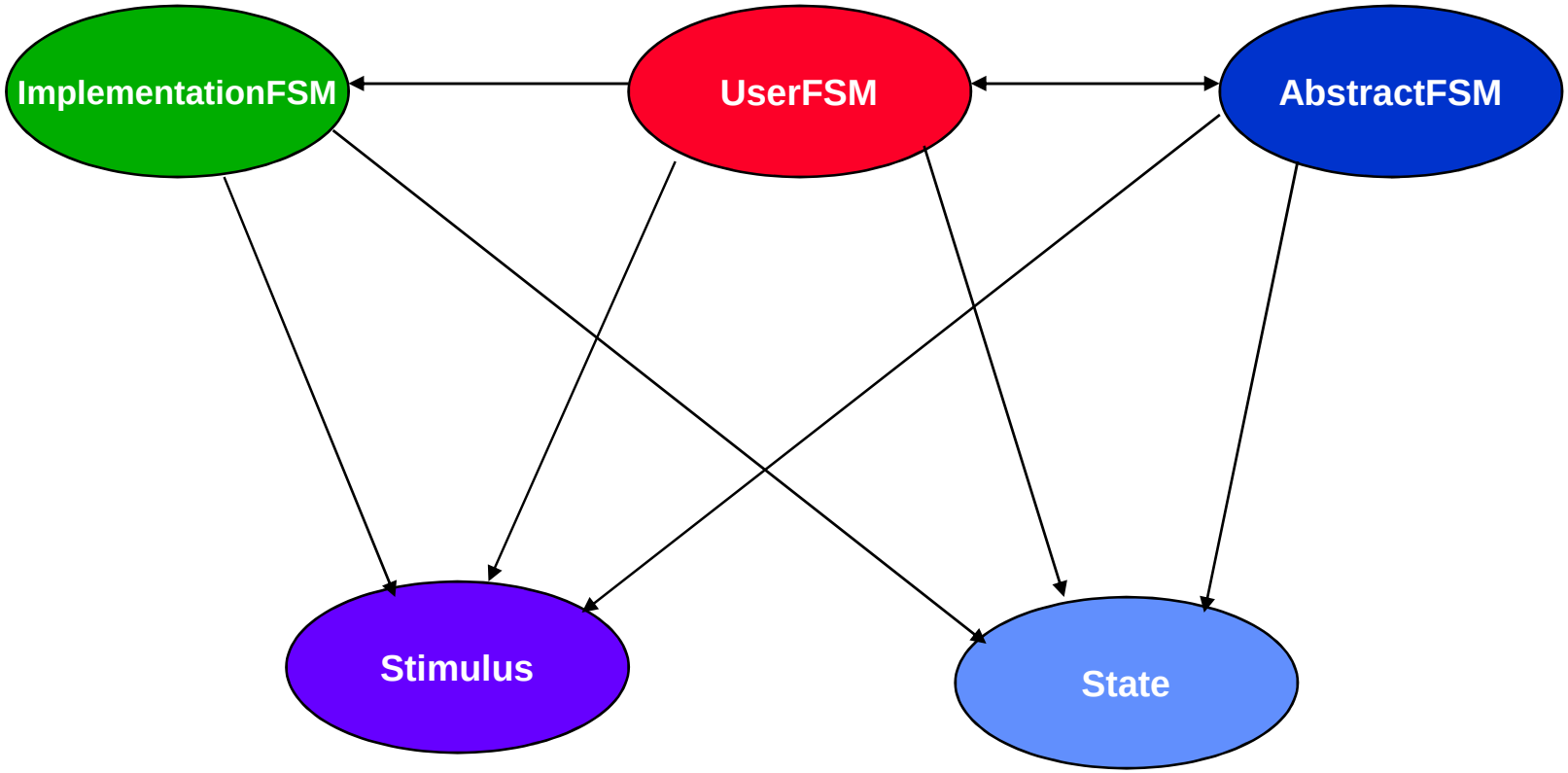
Parameters of Variation	Meaning	Domain	Binding	Default
UserFSM *Structure, algorithm*	To implement the state/action map, the implementation must know the type of user-defined actions and transitions	See previous slide	Compile time	None *Templates*
State *Type*	How to represent the FSM state	Any discrete type	Compile time	None *Templates*
Stimulus *Type*	The type of the message that sequences the machine between states	Any discrete type	Compile time	None *Templates*

UserFSM 可変性テーブル

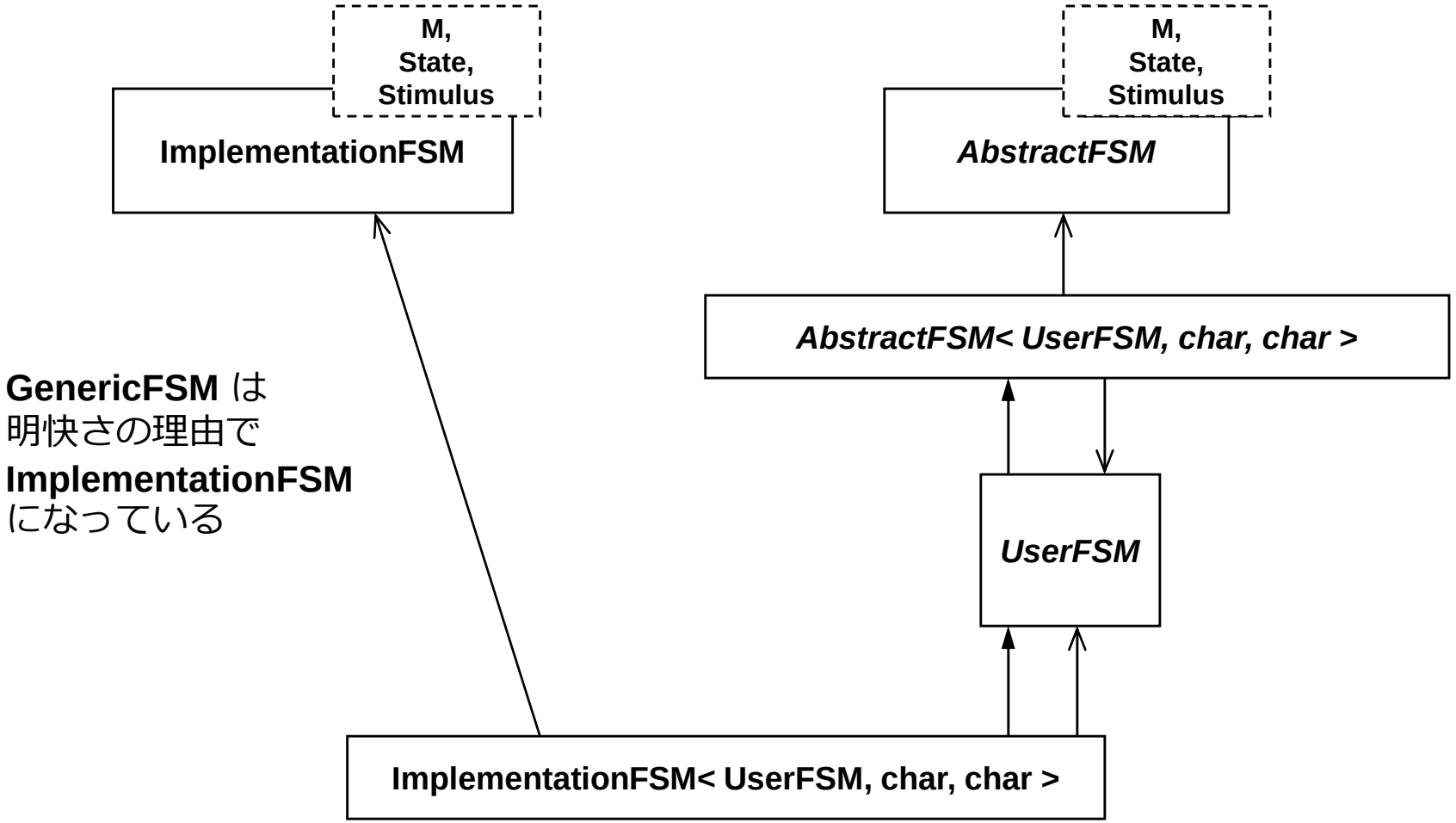
共通性: トータルとしての振る舞い(Aggregate Behavior)

Parameters of Variation	Meaning	Domain	Binding	Default
AbstractFSM *Structure, algorithm*	The UserFSM uses the protocol from the AbstractFSM domain	See previous slide	Compile time	None *Inheritance*
State *Type*	How to represent the FSM state	Any discrete type	Compile time	None *Hand-coded or typedef*
Stimulus *Type*	The type of the message that sequences the machine between states	Any discrete type	Compile time	None *Hand-coded or typedef*
Actions *Algorithm*	Each UserFSM implements its own semantics in transition functions	Any number of functions that map a Stimulus and current state to a new state	Compile time	None *Inheritance*

FSM のドメインダイアグラム



FSMをUML で表現する



GenericFSM は
明快さの理由で
ImplementationFSM
になっている

FSM設計に対応するコード

```
#include <Map.h>

template<class M, class State, class Stimulus>
struct AbstractFSM {
    virtual void addState(State) = 0;
    virtual void addTransition(Stimulus, State, State,
        void (M::*)(Stimulus));
    virtual void fire(Stimulus) = 0;
};

struct MyFSM: public AbstractFSM<MyFSM, char, char> {
    void x1(char);
    void x2(char);
    void init() {
        addState(1);  addState(2);
        addTransition(EOF,2,3,&MyFSM::x1);
    }
};
```

さらに，FSM設計に対応するコード

```
template <class UserMachine, class State, class
  Stimulus>
class FSM: public UserMachine {
public:
    FSM() { init(); }
    virtual void addState(State);
    virtual void addTransition(Stimulus, State
  from,
            State to, void
  (UserMachine::*)(Stimulus));
    virtual void fire(Stimulus);
private:
    State nstates, *states, currentState,
    Map<Stimulus, void (UserMachine::*)(Stimulus)>
        *transitionMap;
};

FSM<MyFSM, char, char> myMachine;
```

オブジェクト，マルチパラダイムデザイン，そしてパターン

- パターン:
  　解決済みの問題に，ソリューションのすでに分かっている組を適用するというテクニック
- OOA/OOD: 抽象を見つけ出すテクニック
- マルチパラダイムデザイン:
  　抽象化のテクニックを見い出し，それを現実的なものに還元する手法