US 20070240110A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2007/0240110 A1**

Rojer (43) **Pub. Date: Oct. 11, 2007**

(54) **PROCESSING A MODULE SPECIFICATION TO PRODUCE A MODULE DEFINITION**

(76) Inventor: **Alan S. Rojer**, Maplewood, NJ (US)

Correspondence Address:
DIMELAB, LLC
BOX 336
MAPLEWOOD, NJ 07040-0336

(21) Appl. No.: **11/784,914**

(22) Filed: **Apr. 10, 2007**

Related U.S. Application Data

(60) Provisional application No. 60/791,128, filed on Apr. 11, 2006.

Publication Classification

(51) **Int. Cl.**
G06F 9/44 (2006.01)

(52) **U.S. Cl.** **717/114**; 717/104

(57) **ABSTRACT**

A computer-implemented method of processing a module specification to produce a module definition is disclosed. A module consists of a collection of interrelated classes for object-oriented programming. A module specification is an object-oriented data structure, the elements of which correspond to classes of a meta-module. A module definition is a collection of class definitions and other expressions in an object-oriented programming language. The module specification includes unitary and categorical class specifications. The class specifications include member specifications. The module definition may include definitions for a unitary host class, a unitary viewer class, a unitary editor class, and categorical class definitions corresponding to the categorical class specifications.

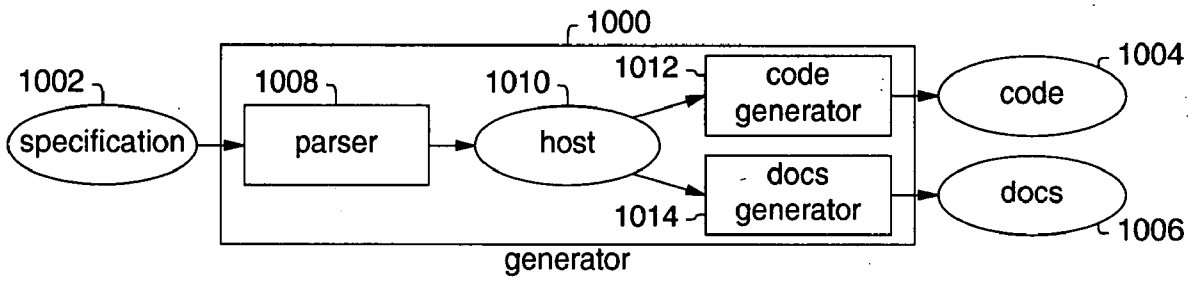

FIG. 1

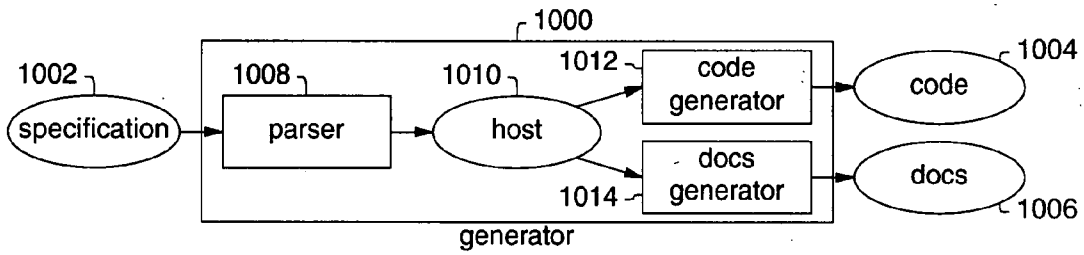

FIG. 2

```
dish[1016] [
  viewer dishViewer[1018];
] {
  dishEntity[1020] {
    Text id[1022];
    dishModule[1024] {
      Sequence<dishClass[1034]> classes[1026] [meron];
      dishHostClass[1040] host[1028] [meron];
      Text viewer_id[1030];
      Text editor_id[1032];
    }
    dishClass[1034] {
      Sequence<dishClass[1034]> genera[1036];
      Sequence<dishMember[1048]> members[1038] [meron];
      dishHostClass[1040] {}
    }
    dishOperand[1042] {
      Text scope_id[1044];
      dishType[1054] type[1046];
      dishMember[1048] {
        dishMemberDatum[1050] {
          Boolean is_meron[1052];
        }
      }
    }
  }
}
```

FIG. 3

```
dish[1016] {
  dishType[1054] {
    dishValueType[1056] {
      dishBitType[1058] {}
      dishTextType[1060] {}
    }
    dishReferenceType[1062] {
      Text reference_class_id[1064];
    }
    dishCompoundType[1066] {
      dishType[1054] range[1068];
      dishSequenceType[1070] {
        dishValueSequenceType[1072] {}
        dishReferenceSequenceType[1074] {}
      }
      dishSetType[1076] {
        dishValueSetType[1078] {}
        dishReferenceSetType[1080] {}
      }
      dishMapType[1082] {
        dishType[1054] domain[1084];
        dishIndexMapType[1086] {}
        dishScaleMapType[1088] {}
        dishBindMapType[1090] {}
        dishConvertMapType[1092] {}
      }
    }
  }
}
```

FIG. 4

```
class dishViewer[1018] {
public:
  /* Class Views[1094] */
  /* Meron View Dispatches[1096] */
  /* Base View Dispatches[1098] */
  /* Aggregate View Dispatches[1100] */
public: // failure state and action...
  bool failed[1102];
  bool fail[1104](Text msg) {
    /* report msg */;
    failed[1102] = 1;
    return 0; }
};
```

FIG. 5

```
class dishEntity[1020] {
public: // features...
  Text id[1022];
public: // dispatch to viewer...
  virtual bool dispatch_view[1106](dishViewer[1018] &v) const {
    return v.view[1108](this); }
};
```

FIG. 6

```
/* dishEntity[1020] segment of */ class dishViewer[1018] {
public:
  virtual bool view[1108](const dishEntity[1020] *e) {
    return view_merons[1110](e) && view_base[1112](e); }
  bool view_merons[1110](const dishEntity[1020] *e) {return 1;}
  bool view_base[1112](const dishEntity[1020] *e) {return 1;}
};
```

FIG. 7

```
class dishModule[1024] : public dishEntity[1020] {
public: // features...
  Sequence<dishClass[1034]*> classes[1026];
  dishHostClass[1040]* host[1028];
  Text viewer_id[1030];
  Text editor_id[1032];
public: // dispatch to viewer...
  virtual bool dispatch_view[1114](dishViewer[1018] &v) const {
    return v.view[1116](this); }
public: // cleanup...
  virtual ~dishModule[1024]() {
    /* cleanup classes[1026] */
    /* cleanup host[1028] */ }
};
```

FIG. 8

```
/* dishModule[1024] segment of */ class dishViewer[1018] {
public:
  virtual bool view[1116](const dishModule[1024] *e) {
    return view_merons[1118](e) && view_base[1120](e); }
  bool view_merons[1118](const dishModule[1024] *e) {
    if (!view_agg[1122](e->classes[1026])) return 0;
    return 1; }
  bool view_base[1120](const dishModule[1024] *e) {
    return e->dishEntity[1020]::dispatch_view[1106](*this); }
  bool view_agg[1122](const Sequence<dishClass[1034]> &s) {
    foreach (const dishClass[1034] *e in s)
      if (!e->dispatch_view[1124](*this)) return 0;
    return 1; }
};
```

FIG. 9

```
class dishClass[1034] : public dishEntity[1020] {
public: // features...
  Sequence<dishClass[1034]*> genera[1036];
  Sequence<dishMember[1048]*> members[1038];
public: // dispatch to viewer...
  virtual bool dispatch_view[1124](dishViewer[1018] &v) const {
    return v.view[1126](this); }
public: // cleanup...
  virtual ~dishClass[1034]() {
    /* cleanup members[1038] */ }
};
```

FIG. 10

```
/* dishClass[1034] segment of */ class dishViewer[1018] {
public:
  virtual bool view[1126](const dishClass[1034] *e) {
    return view_merons[1128](e) && view_base[1130](e); }
  bool view_merons[1128](const dishClass[1034] *e) {
    if (!view_agg[1132](e->members[1038])) return 0;
    return 1; }
  bool view_base[1130](const dishClass[1034] *e) {
    return e->dishEntity[1020]::dispatch_view[1106](*this); }
  bool view_agg[1132](const Sequence<dishMember[1048]> &s) {
    foreach (const dishMember[1048] *e in s)
      if (!e->dispatch_view[1134](*this)) return 0;
    return 1; }
};
```

FIG. 11

```
class dishHostClass[1040] : public dishClass[1034] {
public: // dispatch to viewer...
  virtual bool dispatch_view[1136](dishViewer[1018] &v) const {
    return v.view[1138](this); }
};
```

FIG. 12

```
/* dishHostClass[1040] segment of */ class dishViewer[1018] {
public:
  virtual bool view[1138](const dishHostClass[1040] *e) {
    return view_merons[1140](e) && view_base[1142](e); }
  bool view_merons[1140](const dishHostClass[1040] *e) {return 1;}
  bool view_base[1142](const dishHostClass[1040] *e) {
    return e->dishClass[1034]::dispatch_view[1124](*this); }
};
```

FIG. 13

```
class dishOperand[1042] : public dishEntity[1020] {
public: // features...
  Text scope_id[1044];
  dishType[1054]* type[1046];
public: // dispatch to viewer...
  virtual bool dispatch_view[1144](dishViewer[1018] &v) const {
    return v.view[1146](this); }
};
```

FIG. 14

```
/* dishOperand[1042] segment of */ class dishViewer[1018] {
public:
  virtual bool view[1146](const dishOperand[1042] *e) {
    return view_merons[1148](e) && view_base[1150](e); }
  bool view_merons[1148](const dishOperand[1042] *e) {return 1;}
  bool view_base[1150](const dishOperand[1042] *e) {
    return e->dishEntity[1020]::dispatch_view[1106](*this); }
};
```

FIG. 15

```
class dishMember[1048] : public dishOperand[1042] {
public: // dispatch to viewer...
  virtual bool dispatch_view[1134](dishViewer[1018] &v) const {
    return v.view[1152](this); }
};
```

FIG. 16

```
/* dishMember[1048] segment of */ class dishViewer[1018] {
public:
  virtual bool view[1152](const dishMember[1048] *e) {
    return view_merons[1154](e) && view_base[1156](e); }
  bool view_merons[1154](const dishMember[1048] *e) {return 1;}
  bool view_base[1156](const dishMember[1048] *e) {
    return e->dishOperand[1042]::dispatch_view[1144](*this); }
};
```

FIG. 17

```
class dishMemberDatum[1050] : public dishMember[1048] {
public: // features...
  Boolean is_meron[1052];
public: // dispatch to viewer...
  virtual bool dispatch_view[1158](dishViewer[1018] &v) const {
    return v.view[1160](this); }
};
```

FIG. 18

```
/* dishMemberDatum[1050] segment of */ class dishViewer[1018] {
public:
  virtual bool view[1160](const dishMemberDatum[1050] *e) {
    return view_merons[1162](e) && view_base[1164](e); }
  bool view_merons[1162](const dishMemberDatum[1050] *e) {return 1;}
  bool view_base[1164](const dishMemberDatum[1050] *e) {
    return e->dishMember[1048]::dispatch_view[1134](*this); }
};
```

FIG. 19

```
class dishType[1054] {
public: // dispatch to viewer...
  virtual bool dispatch_view[1166](dishViewer[1018] &v) const {
    return v.view[1168](this); }
};
```

FIG. 20

```
/* dishType[1054] segment of */ class dishViewer[1018] {
public:
  virtual bool view[1168](const dishType[1054] *e) {
    return view_merons[1170](e) && view_base[1172](e); }
  bool view_merons[1170](const dishType[1054] *e) {return 1;}
  bool view_base[1172](const dishType[1054] *e) {return 1;}
};
```

FIG. 21

```
class dishValueType1056 : public dishType1054 {
public: // dispatch to viewer...
  virtual bool dispatch_view1174(dishViewer1018 &v) const {
    return v.view1176(this); }
};
```

FIG. 22

```
/* dishValueType1056 segment of */ class dishViewer1018 {
public:
  virtual bool view1176(const dishValueType1056 *e) {
    return view_merons1178(e) && view_base1180(e); }
  bool view_merons1178(const dishValueType1056 *e) {return 1;}
  bool view_base1180(const dishValueType1056 *e) {
    return e->dishType1054::dispatch_view1166(*this); }
};
```

FIG. 23

```
class dishBitType1058 : public dishValueType1056 {
public: // dispatch to viewer...
  virtual bool dispatch_view1182(dishViewer1018 &v) const {
    return v.view1184(this); }
};
```

FIG. 24

```
/* dishBitType1058 segment of */ class dishViewer1018 {
public:
  virtual bool view1184(const dishBitType1058 *e) {
    return view_merons1186(e) && view_base1188(e); }
  bool view_merons1186(const dishBitType1058 *e) {return 1;}
  bool view_base1188(const dishBitType1058 *e) {
    return e->dishValueType1056::dispatch_view1174(*this); }
};
```

FIG. 25

```
class dishTextType[1060] : public dishValueType[1056] {
public: // dispatch to viewer...
  virtual bool dispatch_view[1190](dishViewer[1018] &v) const {
    return v.view[1192](this); }
};
```

FIG. 26

```
/* dishTextType[1060] segment of */ class dishViewer[1018] {
public:
  virtual bool view[1192](const dishTextType[1060] *e) {
    return view_merons[1194](e) && view_base[1196](e); }
  bool view_merons[1194](const dishTextType[1060] *e) {return 1;}
  bool view_base[1196](const dishTextType[1060] *e) {
    return e->dishValueType[1056]::dispatch_view[1174](*this); }
};
```

FIG. 27

```
class dishReferenceType[1062] : public dishType[1054] {
public: // features...
  Text reference_class_id[1064];
public: // dispatch to viewer...
  virtual bool dispatch_view[1198](dishViewer[1018] &v) const {
    return v.view[1200](this); }
};
```

FIG. 28

```
/* dishReferenceType[1062] segment of */ class dishViewer[1018] {
public:
  virtual bool view[1200](const dishReferenceType[1062] *e) {
    return view_merons[1202](e) && view_base[1204](e); }
  bool view_merons[1202](const dishReferenceType[1062] *e) {return 1;}
  bool view_base[1204](const dishReferenceType[1062] *e) {
    return e->dishType[1054]::dispatch_view[1166](*this); }
};
```

FIG. 29

```
class dishCompoundType1066 : public dishType1054 {
public: // features...
  dishType1054* range1068;
public: // dispatch to viewer...
  virtual bool dispatch_view1206(dishViewer1018 &v) const {
    return v.view1208(this); }
};
```

FIG. 30

```
/* dishCompoundType1066 segment of */ class dishViewer1018 {
public:
  virtual bool view1208(const dishCompoundType1066 *e) {
    return view_merons1210(e) && view_base1212(e); }
  bool view_merons1210(const dishCompoundType1066 *e) {return 1;}
  bool view_base1212(const dishCompoundType1066 *e) {
    return e->dishType1054::dispatch_view1166(*this); }
};
```

FIG. 31

```
class dishSequenceType1070 : public dishCompoundType1066 {
public: // dispatch to viewer...
  virtual bool dispatch_view1214(dishViewer1018 &v) const {
    return v.view1216(this); }
};
```

FIG. 32

```
/* dishSequenceType1070 segment of */ class dishViewer1018 {
public:
  virtual bool view1216(const dishSequenceType1070 *e) {
    return view_merons1218(e) && view_base1220(e); }
  bool view_merons1218(const dishSequenceType1070 *e) {return 1;}
  bool view_base1220(const dishSequenceType1070 *e) {
    return e->dishCompoundType1066::dispatch_view1206(*this); }
};
```

FIG. 33

```
class dishValueSequenceType[1072] : public dishSequenceType[1070] {
public: // dispatch to viewer...
  virtual bool dispatch_view[1222](dishViewer[1018] &v) const {
    return v.view[1224](this); }
};
```

FIG. 34

```
/* dishValueSequenceType[1072] segment of */ class dishViewer[1018] {
public:
  virtual bool view[1224](const dishValueSequenceType[1072] *e) {
    return view_merons[1226](e) && view_base[1228](e); }
  bool view_merons[1226](const dishValueSequenceType[1072] *e) {return 1;}
  bool view_base[1228](const dishValueSequenceType[1072] *e) {
    return e->dishSequenceType[1070]::dispatch_view[1214](*this); }
};
```

FIG. 35

```
class dishReferenceSequenceType[1074] : public dishSequenceType[1070] {
public: // dispatch to viewer...
  virtual bool dispatch_view[1230](dishViewer[1018] &v) const {
    return v.view[1232](this); }
};
```

FIG. 36

```
/* dishReferenceSequenceType[1074] segment of */ class dishViewer[1018] {
public:
  virtual bool view[1232](const dishReferenceSequenceType[1074] *e) {
    return view_merons[1234](e) && view_base[1236](e); }
  bool view_merons[1234](const dishReferenceSequenceType[1074] *e) {return 1;}
  bool view_base[1236](const dishReferenceSequenceType[1074] *e) {
    return e->dishSequenceType[1070]::dispatch_view[1214](*this); }
};
```

FIG. 37

```
class dishSetType[1076] : public dishCompoundType[1066] {
public: // dispatch to viewer...
  virtual bool dispatch_view[1238](dishViewer[1018] &v) const {
    return v.view[1240](this); }
};
```

FIG. 38

```
/* dishSetType[1076] segment of */ class dishViewer[1018] {
public:
  virtual bool view[1240](const dishSetType[1076] *e) {
    return view_merons[1242](e) && view_base[1244](e); }
  bool view_merons[1242](const dishSetType[1076] *e) {return 1;}
  bool view_base[1244](const dishSetType[1076] *e) {
    return e->dishCompoundType[1066]::dispatch_view[1206](*this); }
};
```

FIG. 39

```
class dishValueSetType[1078] : public dishSetType[1076] {
public: // dispatch to viewer...
  virtual bool dispatch_view[1246](dishViewer[1018] &v) const {
    return v.view[1248](this); }
};
```

FIG. 40

```
/* dishValueSetType[1078] segment of */ class dishViewer[1018] {
public:
  virtual bool view[1248](const dishValueSetType[1078] *e) {
    return view_merons[1250](e) && view_base[1252](e); }
  bool view_merons[1250](const dishValueSetType[1078] *e) {return 1;}
  bool view_base[1252](const dishValueSetType[1078] *e) {
    return e->dishSetType[1076]::dispatch_view[1238](*this); }
};
```

FIG. 41

```
class dishReferenceSetType[1080] : public dishSetType[1076] {
public: // dispatch to viewer...
  virtual bool dispatch_view[1254](dishViewer[1018] &v) const {
    return v.view[1256](this); }
};
```

FIG. 42

```
/* dishReferenceSetType[1080] segment of */ class dishViewer[1018] {
public:
  virtual bool view[1256](const dishReferenceSetType[1080] *e) {
    return view_merons[1258](e) && view_base[1260](e); }
  bool view_merons[1258](const dishReferenceSetType[1080] *e) {return 1;}
  bool view_base[1260](const dishReferenceSetType[1080] *e) {
    return e->dishSetType[1076]::dispatch_view[1238](*this); }
};
```

FIG. 43

```
class dishMapType[1082] : public dishCompoundType[1066] {
public: // features...
  dishType[1054]* domain[1084];
public: // dispatch to viewer...
  virtual bool dispatch_view[1262](dishViewer[1018] &v) const {
    return v.view[1264](this); }
};
```

FIG. 44

```
/* dishMapType[1082] segment of */ class dishViewer[1018] {
public:
  virtual bool view[1264](const dishMapType[1082] *e) {
    return view_merons[1266](e) && view_base[1268](e); }
  bool view_merons[1266](const dishMapType[1082] *e) {return 1;}
  bool view_base[1268](const dishMapType[1082] *e) {
    return e->dishCompoundType[1066]::dispatch_view[1206](*this); }
};
```

FIG. 45

```
class dishIndexMapType1086 : public dishMapType1082 {
public: // dispatch to viewer...
  virtual bool dispatch_view1270(dishViewer1018 &v) const {
    return v.view1272(this); }
};
```

FIG. 46

```
/* dishIndexMapType1086 segment of */ class dishViewer1018 {
public:
  virtual bool view1272(const dishIndexMapType1086 *e) {
    return view_merons1274(e) && view_base1276(e); }
  bool view_merons1274(const dishIndexMapType1086 *e) {return 1;}
  bool view_base1276(const dishIndexMapType1086 *e) {
    return e->dishMapType1082::dispatch_view1262(*this); }
};
```

FIG. 47

```
class dishScaleMapType1088 : public dishMapType1082 {
public: // dispatch to viewer...
  virtual bool dispatch_view1278(dishViewer1018 &v) const {
    return v.view1280(this); }
};
```

FIG. 48

```
/* dishScaleMapType1088 segment of */ class dishViewer1018 {
public:
  virtual bool view1280(const dishScaleMapType1088 *e) {
    return view_merons1282(e) && view_base1284(e); }
  bool view_merons1282(const dishScaleMapType1088 *e) {return 1;}
  bool view_base1284(const dishScaleMapType1088 *e) {
    return e->dishMapType1082::dispatch_view1262(*this); }
};
```

FIG. 49

```
class dishBindMapType[1090] : public dishMapType[1082] {
public: // dispatch to viewer...
  virtual bool dispatch_view[1286](dishViewer[1018] &v) const {
    return v.view[1288](this); }
};
```

FIG. 50

```
/* dishBindMapType[1090] segment of */ class dishViewer[1018] {
public:
  virtual bool view[1288](const dishBindMapType[1090] *e) {
    return view_merons[1290](e) && view_base[1292](e); }
  bool view_merons[1290](const dishBindMapType[1090] *e) {return 1;}
  bool view_base[1292](const dishBindMapType[1090] *e) {
    return e->dishMapType[1082]::dispatch_view[1262](*this); }
};
```

FIG. 51

```
class dishConvertMapType[1092] : public dishMapType[1082] {
public: // dispatch to viewer...
  virtual bool dispatch_view[1294](dishViewer[1018] &v) const {
    return v.view[1296](this); }
};
```

FIG. 52

```
/* dishConvertMapType[1092] segment of */ class dishViewer[1018] {
public:
  virtual bool view[1296](const dishConvertMapType[1092] *e) {
    return view_merons[1298](e) && view_base[1300](e); }
  bool view_merons[1298](const dishConvertMapType[1092] *e) {return 1;}
  bool view_base[1300](const dishConvertMapType[1092] *e) {
    return e->dishMapType[1082]::dispatch_view[1262](*this); }
};
```

FIG. 53

```
class dishTypeDefiner[1302] : public dishViewer[1018] {
public: // data...
  Text type_id[1304];
  Text def[1306];
  Text typedef[1308];
  Text include[1310];
  Text template[1312];
public: // entries and categorical views...
  dishTypeDefiner[1314](const dishType[1054]*);
  virtual bool view[1320](const dishBitType[1058]*);
  virtual bool view[1328](const dishTextType[1060]*);
  virtual bool view[1336](const dishReferenceType[1062]*);
  virtual bool view[1346](const dishCompoundType[1066]*);
  virtual bool view[1354](const dishSequenceType[1070]*);
  virtual bool view[1370](const dishReferenceSequenceType[1074]*);
  virtual bool view[1380](const dishValueSequenceType[1072]*);
  virtual bool view[1390](const dishSetType[1076]*);
  virtual bool view[1406](const dishReferenceSetType[1080]*);
  virtual bool view[1416](const dishValueSetType[1078]*);
  virtual bool view[1426](const dishMapType[1082]*);
  virtual bool view[1444](const dishIndexMapType[1086]*);
  virtual bool view[1454](const dishScaleMapType[1088]*);
  virtual bool view[1464](const dishBindMapType[1090]*);
  virtual bool view[1474](const dishConvertMapType[1092]*);
};
```

FIG. 54

```
dishTypeDefiner[1302]::dishTypeDefiner[1314](
  const dishType[1054]* type[1316]
) {
  /* 1318: */ type[1316]->dispatch_view[1166](*this);
}
```

FIG. 55

```
bool dishTypeDefiner[1302]::view[1320](
  const dishBitType[1058]* bit_type[1322]
) {
  /* 1324: */ type_id[1304] << "bool";
  /* 1326: */ return 1;
}
```

FIG. 56

```
bool dishTypeDefiner1302::view1328(
  const dishTextType1060* text_type1330
) {
  /* 1332: */ type_id1304 << "Text";
  /* 1334: */ return 1;
}
```

FIG. 57

```
bool dishTypeDefiner1302::view1336(
  const dishReferenceType1062* reference_type1338
) {
  /* 1340: */ type_id1304 << reference_type1338->reference_class_id1064;
  /* 1342: */ def1306.printf("%s*", reference_type1338->reference_class_id1064);
  /* 1344: */ return 1;
}
```

FIG. 58

```
bool dishTypeDefiner1302::view1346(
  const dishCompoundType1066* compound_type1348
) {
  /* 1350: */ typedef1308.printf("typedef %s %s", type_id1304, def1306);
  /* 1352: */ return 1;
}
```

FIG. 59

```
bool dishTypeDefiner1302::view1354(
  const dishSequenceType1070* sequence_type1356
) {
  /* 1358: */ include1310 << "include Sequence.h";
  /* 1360: */ dishTypeDefiner1302 range_formatter(sequence_type1356->range1068);
  /* 1362: */ def1306.printf("%s<%s>", range_formatter.type_id1304, template1312);
  /* 1364: */ type_id1304.printf("%sSequence", range_formatter.type_id1304);
  /* 1366: */ view_base1220(sequence_type1356);
  /* 1368: */ return 1;
}
```

FIG. 60

```
bool dishTypeDefiner1302::view1370(
  const dishReferenceSequenceType1074* reference_sequence_type1372
) {
  /* 1374: */ template1312 << "PointerSequence";
  /* 1376: */ view_base1236(reference_sequence_type1372);
  /* 1378: */ return 1;
}
```

FIG. 61

```
bool dishTypeDefiner1302::view1380(
  const dishValueSequenceType1072* value_sequence_type1382
) {
  /* 1384: */ template1312 << "Sequence";
  /* 1386: */ view_base1228(value_sequence_type1382);
  /* 1388: */ return 1;
}
```

FIG. 62

```
bool dishTypeDefiner1302::view1390(
  const dishSetType1076* set_type1392
) {
  /* 1394: */ include1310 << "include Set.h";
  /* 1396: */ dishTypeDefiner1302 range_formatter(set_type1392->range1068);
  /* 1398: */ def1306.printf("%s<%s>", range_formatter.type_id1304, template1312);
  /* 1400: */ type_id1304.printf("%sSet", range_formatter.type_id1304);
  /* 1402: */ view_base1244(set_type1392);
  /* 1404: */ return 1;
}
```

FIG. 63

```
bool dishTypeDefiner1302::view1406(
  const dishReferenceSetType1080* reference_set_type1408
) {
  /* 1410: */ template1312 << "PointerSet";
  /* 1412: */ view_base1260(reference_set_type1408);
  /* 1414: */ return 1;
}
```

FIG. 64

```
bool dishTypeDefiner1302::view1416(
  const dishValueSetType1078* value_set_type1418
) {
  /* 1420: */ template1312 << "Set";
  /* 1422: */ view_base1252(value_set_type1418);
  /* 1424: */ return 1;
}
```

FIG. 65

```
bool dishTypeDefiner1302::view1426(
  const dishMapType1082* map_type1428
) {
  /* 1430: */ include1310 << "include Map.h";
  /* 1432: */ dishTypeDefiner1302 range_formatter(map_type1428->range1068);
  /* 1434: */ dishTypeDefiner1302 domain_formatter(map_type1428->domain1084);
  /* 1436: */ def1306.printf(
    "%s<%s, %s>",
    range_formatter.type_id1304,
    domain_formatter.type_id1304,
    template1312);
  /* 1438: */ type_id1304.printf(
    "%s%sSet",
    range_formatter.type_id1304,
    domain_formatter.type_id1304,
    template1312);
  /* 1440: */ view_base1268(map_type1428);
  /* 1442: */ return 1;
}
```

FIG. 66

```
bool dishTypeDefiner1302::view1444(
  const dishIndexMapType1086* index_map_type1446
) {
  /* 1448: */ template1312 << "Index";
  /* 1450: */ view_base1276(index_map_type1446);
  /* 1452: */ return 1;
}
```

FIG. 67

```
bool dishTypeDefiner1302::view1454(
  const dishScaleMapType1088* scale_map_type1456
) {
  /* 1458: */ template1312 << "Scale";
  /* 1460: */ view_base1284(scale_map_type1456);
  /* 1462: */ return 1;
}
```

FIG. 68

```
bool dishTypeDefiner1302::view1464(
  const dishBindMapType1090* bind_map_type1466
) {
  /* 1468: */ template1312 << "Bind";
  /* 1470: */ view_base1292(bind_map_type1466);
  /* 1472: */ return 1;
}
```

FIG. 69

```
bool dishTypeDefiner1302::view1474(
  const dishConvertMapType1092* convert_map_type1476
) {
  /* 1478: */ template1312 << "Convert";
  /* 1480: */ view_base1300(convert_map_type1476);
  /* 1482: */ return 1;
}
```

FIG. 70

```
class dishForwardsWriter[1484] : public dishViewer[1018] {
public: // data...
  Text viewer_forward[1486];
  Text forwards[1488];
  Text editor_forward[1490];
  Text cat_forward[1492];
  Set<Text> includes[1494];
  Set<Text> typedefs[1496];
  Text host_forward[1498];
public: // entries and categorical views...
  dishForwardsWriter[1500](const dishModule[1024]*);
  virtual bool view[1516](const dishClass[1034]*);
  virtual bool view[1528](const dishMemberDatum[1050]*);
  virtual bool view[1540](const dishHostClass[1040]*);
};
```

FIG. 71

```
dishForwardsWriter[1484]::dishForwardsWriter[1500](
  const dishModule[1024]* module[1502]
) {
  /* 1504: */ view_agg[1122](module[1502]->classes[1026]);
  /* 1506: */ module[1502]->host[1028]->dispatch_view[1136](*this);
  /* 1508: */ viewer_forward[1486].printf("class %s;", module[1502]->viewer_id[1030]);
  /* 1510: */ forwards[1488] << viewer_forward[1486];
  /* 1512: */ editor_forward[1490].printf("class %s;", module[1502]->editor_id[1032]);
  /* 1514: */ forwards[1488] << editor_forward[1490];
}
```

FIG. 72

```
bool dishForwardsWriter[1484]::view[1516](
  const dishClass[1034]* class[1518]
) {
  /* 1520: */ cat_forward[1492].printf("class %s;", class[1518]->id[1022]);
  /* 1522: */ forwards[1488] << cat_forward[1492];
  /* 1524: */ view_agg[1132](class[1518]->members[1038]);
  /* 1526: */ return 1;
}
```

FIG. 73

```
bool dishForwardsWriter1484::view1528(
  const dishMemberDatum1050* member_datum1530
) {
  /* 1532: */ dishTypeDefiner1302 type_definer(member_datum1530->type1046);
  /* 1534: */ includes1494 << type_definer.include1310;
  /* 1536: */ typedefs1496 << type_definer.typedef1308;
  /* 1538: */ return 1;
}
```

FIG. 74

```
bool dishForwardsWriter1484::view1540(
  const dishHostClass1040* host_class1542
) {
  /* 1544: */ host_forward1498.printf("class %s;", host_class1542->id1022);
  /* 1546: */ forwards1488 << host_forward1498;
  /* 1548: */ return 1;
}
```

FIG. 75

```
class dishModuleDatumWriter1550 : public dishViewer1018 {
public: // data...
  Text datum_def1552;
  Text initializer1554;
  Text cleanup1556;
  Text meron_view_dispatch1558;
  Text meron_edit_dispatch1560;
  Text view_access_def1562;
  Text view_access_defs1564;
  Text reset_access_def1566;
  Text reset_access_defs1568;
  Text edit_access_def1570;
  Text edit_access_defs1572;
public: // entries and categorical views...
  dishModuleDatumWriter1574(const dishMemberDatum1050*);
  virtual bool view1584(const dishValueType1056*);
  virtual bool view1598(const dishReferenceType1062*);
  virtual bool view1628(const dishCompoundType1066*);
  virtual bool view1646(const dishReferenceSequenceType1074*);
  virtual bool view1658(const dishReferenceSetType1080*);
};
```

FIG. 76

```
dishModuleDatumWriter1550::dishModuleDatumWriter1574(
  const dishMemberDatum1050* member_datum1576
) {
  /* 1578: */ dishTypeDefiner1302 type_definer(member_datum1576->type1046);
  /* 1580: */ datum_def1552.printf(
    "%s %s",
    member_datum1576->scope_id1044,
    type_definer.type_id1304);
  /* 1582: */ member_datum1576->type1046->dispatch_view1166(*this);
}
```

FIG. 77

```
bool dishModuleDatumWriter1550::view1584(
  const dishValueType1056* value_type1586
) {
  /* 1588: */ view_access_def1562.printf(
    "%s %s() const",
    datum->scope_id1044,
    type_definer.type_id1304);
  /* 1590: */ view_access_defs1564 << view_access_def1562;
  /* 1592: */ reset_access_def1566.printf(
    "void set_%s(%s)",
    datum->scope_id1044,
    type_definer.type_id1304);
  /* 1594: */ reset_access_defs1568 << reset_access_def1566;
  /* 1596: */ return 1;
}
```

FIG. 78

```
bool dishModuleDatumWriter1550::view1598(
  const dishReferenceType1062* reference_type1600
) {
  /* 1602: */ view_access_def1562.printf(
    "const %s* %s() const",
    datum->scope_id1044,
    type_definer.type_id1304);
  /* 1604: */ view_access_defs1564 << view_access_def1562;
  /* 1606: */ edit_access_def1570.printf(
    "%s* %s(%s)",
    datum->scope_id1044,
    type_definer.type_id1304);
  /* 1608: */ edit_access_defs1572 << edit_access_def1570;
  /* 1610: */ reset_access_def1566.printf(
    "void set_%s(%s*)",
    datum->scope_id1044,
    type_definer.type_id1304);
  /* 1612: */ initializer1554 << "0";
  /* 1614: */ if (datum->is_meron1052) {
    /* 1616: */ cleanup1556.printf("delete %s;", datum->scope_id1044);
    /* 1618: */ reset_access_def1566.printf(
      "delete %s;",
      datum->scope_id1044);
    /* 1620: */ meron_view_dispatch1558.printf(
      "%s->dispatch_view(viewer)",
      datum->scope_id1044);
    /* 1622: */ meron_edit_dispatch1560.printf(
      "%s->dispatch_edit(viewer)",
      datum->scope_id1044);
  }
  /* 1624: */ reset_access_defs1568 << reset_access_def1566;
  /* 1626: */ return 1;
}
```

FIG. 79

```
bool dishModuleDatumWriter1550::view1628(
  const dishCompoundType1066* compound_type1630
) {
  /* 1632: */ view_access_def1562.printf(
    "const %s& %s() const",
    datum->scope_id1044,
    type_definer.type_id1304);
  /* 1634: */ view_access_defs1564 << view_access_def1562;
  /* 1636: */ edit_access_def1570.printf(
    "%s& %s(%s)",
    datum->scope_id1044,
    type_definer.type_id1304);
  /* 1638: */ edit_access_defs1572 << edit_access_def1570;
  /* 1640: */ if (datum->is_meron1052) {
    /* 1642: */ cleanup1556.printf(
      "/* delete elements of %s */",
      datum->scope_id1044);
  }
  /* 1644: */ return 1;
}
```

FIG. 80

```
bool dishModuleDatumWriter1550::view1646(
  const dishReferenceSequenceType1074* reference_sequence_type1648
) {
  /* 1650: */ if (datum->is_meron1052) {
    /* 1652: */ meron_view_dispatch1558.printf(
      "view_agg(%s)",
      datum->scope_id1044);
    /* 1654: */ meron_edit_dispatch1560.printf(
      "edit_agg(%s)",
      datum->scope_id1044);
  }
  /* 1656: */ return 1;
}
```

FIG. 81

```
bool dishModuleDatumWriter1550::view1658(
  const dishReferenceSetType1080* reference_set_type1660
) {
  /* 1662: */ if (datum->is_meron1052) {
    /* 1664: */ meron_view_dispatch1558.printf(
      "view_agg(%s)",
      datum->scope_id1044);
    /* 1666: */ meron_edit_dispatch1560.printf(
      "edit_agg(%s)",
      datum->scope_id1044);
  }
  /* 1668: */ return 1;
}
```

FIG. 82

```
class dishModuleClassWriter1670 : public dishViewer1018 {
public: // data...
  Text cat_def1672;
  Text class_view1674;
  Text class_edit1676;
  Text meron_view_dispatches1678;
  Text meron_edit_dispatches1680;
  Text datum_defs1682;
  Text base_view_dispatch1684;
  Text base_view_dispatches1686;
  Text view_dispatch1688;
  Text base_edit_dispatch1690;
  Text base_edit_dispatches1692;
  Text edit_dispatch1694;
  Text initializers1696;
  Text cleanups1698;
public: // entries and categorical views...
  dishModuleClassWriter1700(const dishClass1034*);
  virtual bool view1736(const dishMemberDatum1050*);
};
```

FIG. 83

```
dishModuleClassWriter1670::dishModuleClassWriter1700(
  const dishClass1034* class1702
) {
  /* 1704: */ view_agg1132(class1702->members1038);
  /* 1706: */ class_view1674 << meron_view_dispatches1678;
  /* 1708: */ class_edit1676 << meron_edit_dispatches1680;
  /* 1710: */ cat_def1672 << datum_defs1682;
  /* 1712: */ foreach (genus in class1702->genera1036) {
    /* 1714: */ base_view_dispatch1684.printf(
      "e->%s::dispatch_view(*this)",
      genus);
    /* 1716: */ base_view_dispatches1686 << base_view_dispatch1684;
  }
  /* 1718: */ class_view1674 << base_view_dispatches1686;
  /* 1720: */ view_dispatch1688
    << "bool dispatch_view(Visitor& v) const"
    << "{return v.view(this);}";
  /* 1722: */ cat_def1672 << view_dispatch1688;
  /* 1724: */ foreach (genus in class1702->genera1036) {
    /* 1726: */ base_edit_dispatch1690.printf(
      "e->%s::dispatch_edit(*this)",
      genus);
    /* 1728: */ base_edit_dispatches1692 << base_edit_dispatch1690;
  }
  /* 1730: */ class_edit1676 << base_edit_dispatches1692;
  /* 1732: */ edit_dispatch1694
    << "bool dispatch_edit(Visitor& v)"
    << "{return v.edit(this);}";
  /* 1734: */ cat_def1672 << edit_dispatch1694;
}
```

FIG. 84

```
bool dishModuleClassWriter1670::view1736(
  const dishMemberDatum1050* member_datum1738
) {
  /* 1740: */ dishModuleDatumWriter1550 datum_writer(member_datum1738);
  /* 1742: */ datum_defs1682 << datum_writer.datum_def1552;
  /* 1744: */ initializers1696 << datum_writer.initializer1554;
  /* 1746: */ cleanups1698 << datum_writer.cleanup1556;
  /* 1748: */ meron_view_dispatches1678 << datum_writer.meron_view_dispatch1558;
  /* 1750: */ return 1;
}
```

FIG. 85

```
class dishModuleWriter[1752] : public dishViewer[1018] {
public: // data...
  Text module_def[1754];
  Text viewer[1756];
  Text class_views[1758];
  Text editor[1760];
  Text class_edits[1762];
  Text cat_defs[1764];
  Text host_def[1766];
public: // entries and categorical views...
  dishModuleWriter[1768](const dishModule[1024]*);
  virtual bool view[1780](const dishClass[1034]*);
  virtual bool view[1794](const dishHostClass[1040]*);
};
```

FIG. 86

```
dishModuleWriter[1752]::dishModuleWriter[1768](
  const dishModule[1024]* module[1770]
) {
  /* 1772: */ module[1770]->host[1028]->dispatch_view[1136](*this);
  /* 1774: */ view_agg[1122](module[1770]->classes[1026]);
  /* 1776: */ viewer[1756] << class_views[1758];
  /* 1778: */ editor[1760] << class_edits[1762];
}
```

FIG. 87

```
bool dishModuleWriter[1752]::view[1780](
  const dishClass[1034]* class[1782]
) {
  /* 1784: */ dishModuleClassWriter[1670] class_writer(class[1782]);
  /* 1786: */ cat_defs[1764] << class_writer.cat_def[1672];
  /* 1788: */ class_views[1758] << class_writer.class_view[1674];
  /* 1790: */ class_edits[1762] << class_writer.class_edit[1676];
  /* 1792: */ return 1;
}
```

FIG. 88

```
bool dishModuleWriter1752::view1794(
  const dishHostClass1040* host_class1796
) {
  /* 1798: */ dishModuleClassWriter1670 host_class_writer(host_class1796);
  /* 1800: */ host_def1766 << host_class_writer.datum_defs1682;
  /* 1802: */ return 1;
}
```

FIG. 89

```
# example dishModule1024...
mscan1804 [
  host mscanHost1806; # ...example host1028
  viewer mscanViewer1808 # ...specify viewer1756 definition
] {
  # example host1028 member...
  Set<mscanFile1810> _root_files1822 [meron];
    # ...example is_meron1052
  # example classes1026...
  mscanFile1810 {
    # example members1038...
    mscanDirectory1812 _parent1824; # ...example dishReferenceType1062
    Text _name1826; # ...example dishTextType1060
    # example genera1036...
    mscanDirectory1812 { # ...mscanDirectory1812 has genus mscanFile1810
      Set<mscanFile1810> _files1828 [meron];
        # ...example is_meron1052
        # ...example dishReferenceSetType1080
      mscanTopDirectory1814 {}
    }
    mscanRegularFile1816 {
      mscanAudioFile1818 {}
      mscanVideoFile1820 {}
    }
  }
}
```

FIG. 90

```
// example module_def1754...

// example includes1494...
#include "Set.h"

// class forward declarations...
class mscanFile1810;
  class mscanDirectory1812;
    class mscanTopDirectory1814;
  class mscanRegularFile1816;
    class mscanAudioFile1818;
    class mscanVideoFile1820;

class mscanHost1806;
class mscanViewer1808;

// example typedefs1496...
typedef Set<mscanFile*> mscanFileSet;

// example host_def1766...
class mscanHost1806 {
private: // member definitions...
  // example datum_def1552...
  mscanFileSet _root_files1822;
public: // member access...
  // example edit_access_defs1572...
  mscanFileSet& root_files() {return _root_files1822;}
  // example view_access_defs1564...
  const mscanFileSet& root_files() const {return _root_files1822;}
public: // factory...
  mscanHost1806() {}
  virtual ~mscanHost1806() {
    // example cleanups1698...
    // example cleanup1556
    /* delete _root_files1822 contents */
  }
};
```

FIG. 91

```
// example cat_defs1764...
// example cat_def1672...
class mscanFile1810 {
private:
  // example datum_def1552...
  mscanDirectory* _parent1824;
  Text _name1826;
public:
  // example edit_access_defs1572...
  mscanDirectory *parent() {return _parent1824;}
  // example view_access_defs1564...
  const mscanDirectory *parent() const {return _parent1824;}
  Text name() const {return _name1826;}
  // example reset_access_defs1568...
  void set_parent(mscanDirectory *t) {_parent1824 = t;}
  void set_name(Text t) {_name1826 = t;}
public:
  // example view_dispatch1688...
  virtual bool dispatch_view(mscanViewer1808 &v) const {return v.view(this);}
public:
  mscanFile1810() : _parent1824(0) {}
    // ...example initializer1554
    // ...example initializers1696
  virtual ~mscanFile1810() {}
};

// example cat_def1672...
class mscanDirectory1812 : public mscanFile1810 {
private: // example datum_defs1682...
  mscanFileSet _files1828;
public:
  // example edit_access_defs1572...
  mscanFileSet& files() {return _files1828;}
  // example view_access_defs1564...
  const mscanFileSet& files() const {return _files1828;}
public:
  // example view_dispatch1688...
  virtual bool dispatch_view(mscanViewer1808 &v) const {return v.view(this);}
public:
  mscanDirectory1812() {}
  virtual ~mscanDirectory1812 {
    // example cleanups1698...
    // example cleanup1556...
    /* delete _files1828 contents */
  }
};
```

FIG. 92

```
// example viewer[1756]...
class mscanViewer[1808] {
public: // example class_views[1758]...
  // example class_view[1674]...
  virtual bool view(const mscanFile[1810]* x) {
    return view_merons(x) && view_base(x);
  }
  // example class_view[1674]...
  virtual bool view(const mscanDirectory[1812]* x) {
    return view_merons(x) && view_base(x);
  }
  // ...
public:
  bool view_merons(const mscanFile[1810]* x) {return 1;}
  bool view_merons(const mscanDirectory[1812]* x) {
    // example meron_view_dispatches[1678]...
    // example meron_view_dispatch[1558]...
    return view_agg(x->files());
  }
  // ...
public:
  bool view_base(const mscanFile[1810]* x) {return 1;}
  bool view_base(const mscanDirectory[1812]* x) {
    // example base_view_dispatches[1686]...
    return x->mscanFile[1810]::dispatch_view(*this);
  }
  // ...
public:
  bool view_agg(const mscanFileSet& a) {
    foreach(u in a) if (!u->dispatch_view(*this)) return 0;
    return 1;
  }
};
```

PROCESSING A MODULE SPECIFICATION TO PRODUCE A MODULE DEFINITION

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of PPA Ser. No. 60/791,128, Module Generation for Object-Oriented Programming, Atty. Docket #2006-003, filed 2006-04-11 by the present inventor, the disclosure of which is incorporated herein by reference.

BACKGROUND OF THE INVENTION

[0002] This invention relates particularly to the production of module definitions from module specifications, and generally to software development tools generating object-oriented programming language code.

[0003] Object-oriented programming enjoys great popularity among programmers. However, in the development of a complex system, the programmer must attend to a vast collection of details. These details may easily obscure and complicate large-scale considerations of the interactions within and between program elements. Hence it would be beneficial to have the use of tools which reduce complexity by automatically handling various details, which may then be suppressed in favor of concise expressions of large-scale interactions.

[0004] Object-oriented programming tends to focus on classes, since classes are the most important component in object-oriented programming languages. In many applications, however, significant benefits may be gained by considering modules, which include collections of interrelated classes. Modules typically incorporate classes which are closely related to domain-specific categories. Relationships of generalizations and specialization among domain categories may be reflected in derivation relationships between categorical classes. It would be beneficial for modules to also include non-categorical classes which are specialized for hosting and processing data structures composed of instances from the categorical classes. It would also be beneficial to coordinate and systematize categorical and unitary non-categorical classes to enhance developer productivity, further relieving the developer from excessive attention to tedious details.

[0005] The problems of specifying suitable representations have led to enormous efforts in the provision of modeling languages, of which the most prominent may be the Unified Modeling Language (UML). UML is vast and, comprehensive, with a scope that encompasses all aspects of object-oriented programming. The breadth of that scope limits the use of idioms, patterns, and other paradigms that are applicable in a narrower context of specific unitary and categorical classes. It would be beneficial to have methods of processing that were specifically directed to the narrower but still critical problems of the specification, construction, and processing of domain-specific object-oriented data structures.

SUMMARY

[0006] A computer-implemented method of processing a module specification to produce a module definition is disclosed. A module is viewed. The module is included in the module specification. A plurality of categorical classes are dispatched. The categorical classes are included in the module. A plurality of class view definitions are accumulated to a viewer class definition. The viewer class definition is included in the module definition.

[0007] A categorical class is viewed. The categorical class is included in the categorical classes. The categorical class may be associated with one or more genus classes. A plurality of class members are dispatched. The class members are included in the categorical class. A datum is viewed. The datum is included in the class members. A datum definition is written. The datum definition is accumulated to a plurality of member data definitions.

[0008] According to a test for a meron qualification in the datum, a member meron view dispatch operation is written. The member view meron dispatch, if any, is accumulated to a plurality of meron view dispatch operations. The meron view dispatch operations are accumulated to a class view definition. The member data definitions are accumulated to a categorical class definition.

[0009] A genus class is iterated over each of the genus classes, if any. A base class view dispatch operation is written using the genus class. The base class view dispatch operation is accumulated to a plurality of base class view dispatch operations. The base class view dispatch operations, if any, are accumulated to the class view definition.

[0010] A view dispatch is written. The view dispatch is accumulated to the categorical class definition. The categorical class definition is accumulated to a plurality of categorical class definitions. The categorical class definitions are included in the module definition. The class view definition is accumulated to the class view definitions.

BRIEF DESCRIPTION OF DRAWINGS

[0011] FIG. **1** depicts a generator which reads module specifications and writes programming language code and documention corresponding to the specifications.

[0012] FIG. **2** begins the depiction of a summary specification of an exemplary object-oriented module suitable for representation of specifications of object-oriented modules (1/2).

[0013] FIG. **3** concludes the depiction of a summary specification of an exemplary object-oriented module suitable for representation of specifications of object-oriented modules (2/2).

[0014] FIG. **4** depicts a skeletal definition of an exemplary viewer class suitable as a base class for specialization to process a data structure representative of specifications of object-oriented modules.

[0015] FIG. **5** depicts an informal definition of an exemplary class representing entity elements.

[0016] FIG. **6** depicts a segment of a viewer base class pertaining to instances of the entity.

[0017] FIG. **7** depicts an informal definition of an exemplary class representing module elements.

[0018] FIG. **8** depicts a segment of a viewer base class pertaining to instances of the module.

[0019] FIG. **9** depicts an informal definition of an exemplary class representing class elements.

[0020] FIG. **10** depicts a segment of a viewer base class pertaining to instances of the class.

[0021] FIG. **11** depicts an informal definition of an exemplary class representing host-class elements.

[0022] FIG. **12** depicts a segment of a viewer base class pertaining to instances of the host-class.

[0023] FIG. **13** depicts an informal definition of an exemplary class representing operand elements.
[0024] FIG. **14** depicts a segment of a viewer base class pertaining to instances of the operand.
[0025] FIG. **15** depicts an informal definition of an exemplary class representing member elements.
[0026] FIG. **16** depicts a segment of a viewer base class pertaining to instances of the member.
[0027] FIG. **17** depicts an informal definition of an exemplary class representing member-datum elements.
[0028] FIG. **18** depicts a segment of a viewer base class pertaining to instances of the member-datum.
[0029] FIG. **19** depicts an informal definition of an exemplary class representing type elements.
[0030] FIG. **20** depicts a segment of a viewer base class pertaining to instances of the type.
[0031] FIG. **21** depicts an informal definition of an exemplary class representing value-type elements.
[0032] FIG. **22** depicts a segment of a viewer base class pertaining to instances of the value-type.
[0033] FIG. **23** depicts an informal definition of an exemplary class representing bit-type elements.
[0034] FIG. **24** depicts a segment of a viewer base class pertaining to instances of the bit-type.
[0035] FIG. **25** depicts an informal definition of an exemplary class representing text-type elements.
[0036] FIG. **26** depicts a segment of a viewer base class pertaining to instances of the text-type.
[0037] FIG. **27** depicts an informal definition of an exemplary class representing reference-type elements.
[0038] FIG. **28** depicts a segment of a viewer base class pertaining to instances of the reference-type.
[0039] FIG. **29** depicts an informal definition of an exemplary class representing compound-type elements.
[0040] FIG. **30** depicts a segment of a viewer base class pertaining to instances of the compound-type.
[0041] FIG. **31** depicts an informal definition of an exemplary class representing sequence-type elements.
[0042] FIG. **32** depicts a segment of a viewer base class pertaining to instances of the sequence-type.
[0043] FIG. **33** depicts an informal definition of an exemplary class representing value-sequence-type elements.
[0044] FIG. **34** depicts a segment of a viewer base class pertaining to instances of the value-sequence-type.
[0045] FIG. **35** depicts an informal definition of an exemplary class representing reference-sequence-type elements.
[0046] FIG. **36** depicts a segment of a viewer base class pertaining to instances of the reference-sequence-type.
[0047] FIG. **37** depicts an informal definition of an exemplary class representing set-type elements.
[0048] FIG. **38** depicts a segment of a viewer base class pertaining to instances of the set-type.
[0049] FIG. **39** depicts an informal definition of an exemplary class representing value-set-type elements.
[0050] FIG. **40** depicts a segment of a viewer base class pertaining to instances of the value-set-type.
[0051] FIG. **41** depicts an informal definition of an exemplary class representing reference-set-type elements.
[0052] FIG. **42** depicts a segment of a viewer base class pertaining to instances of the reference-set-type.
[0053] FIG. **43** depicts an informal definition of an exemplary class representing map-type elements.
[0054] FIG. **44** depicts a segment of a viewer base class pertaining to instances of the map-type.
[0055] FIG. **45** depicts an informal definition of an exemplary class representing index-map-type elements.
[0056] FIG. **46** depicts a segment of a viewer base class pertaining to instances of the index-map-type.
[0057] FIG. **47** depicts an informal definition of an exemplary class representing scale-map-type elements.
[0058] FIG. **48** depicts a segment of a viewer base class pertaining to instances of the scale-map-type.
[0059] FIG. **49** depicts an informal definition of an exemplary class representing bind-map-type elements.
[0060] FIG. **50** depicts a segment of a viewer base class pertaining to instances of the bind-map-type.
[0061] FIG. **51** depicts an informal definition of an exemplary class representing convert-map-type elements.
[0062] FIG. **52** depicts a segment of a viewer base class pertaining to instances of the convert-map-type.
[0063] FIG. **53** depicts an informal definition of an exemplary class suitable for type definer operations.
[0064] FIG. **54** depicts an exemplary informal definition of a type definer member function type-entry.
[0065] FIG. **55** depicts an exemplary informal definition of a type definer member function view-bit-type.
[0066] FIG. **56** depicts an exemplary informal definition of a type definer member function view-text-type.
[0067] FIG. **57** depicts an exemplary informal definition of a type definer member function view-reference-type.
[0068] FIG. **58** depicts an exemplary informal definition of a type definer member function view-compound-type.
[0069] FIG. **59** depicts an exemplary informal definition of a type definer member function view-sequence-type.
[0070] FIG. **60** depicts an exemplary informal definition of a type definer member function view-reference-sequence-type.
[0071] FIG. **61** depicts an exemplary informal definition of a type definer member function view-value-sequence-type.
[0072] FIG. **62** depicts an exemplary informal definition of a type definer member function view-set-type.
[0073] FIG. **63** depicts an exemplary informal definition of a type definer member function view-reference-set-type.
[0074] FIG. **64** depicts an exemplary informal definition of a type definer member function view-value-set-type.
[0075] FIG. **65** depicts an exemplary informal definition of a type definer member function view-map-type.
[0076] FIG. **66** depicts an exemplary informal definition of a type definer member function view-index-map-type.
[0077] FIG. **67** depicts an exemplary informal definition of a type definer member function view-scale-map-type.
[0078] FIG. **68** depicts an exemplary informal definition of a type definer member function view-bind-map-type.
[0079] FIG. **69** depicts an exemplary informal definition of a type definer member function view-convert-map-type.
[0080] FIG. **70** depicts an informal definition of an exemplary class suitable for forwards writer operations.
[0081] FIG. **71** depicts an exemplary informal definition of a forwards writer member function module-entry.
[0082] FIG. **72** depicts an exemplary informal definition of a forwards writer member function view-class.
[0083] FIG. **73** depicts an exemplary informal definition of a forwards writer member function view-member-datum.
[0084] FIG. **74** depicts an exemplary informal definition of a forwards writer member function view-host-class.
[0085] FIG. **75** depicts an informal definition of an exemplary class suitable for member datum writer operations.

[0086] FIG. **76** depicts an exemplary informal definition of a member datum writer member function member-datum-entry.

[0087] FIG. **77** depicts an exemplary informal definition of a member datum writer member function view-value-type.

[0088] FIG. **78** depicts an exemplary informal definition of a member datum writer member function view-reference-type.

[0089] FIG. **79** depicts an exemplary informal definition of a member datum writer member function view-compound-type.

[0090] FIG. **80** depicts an exemplary informal definition of a member datum writer member function view-reference-sequence-type.

[0091] FIG. **81** depicts an exemplary informal definition of a member datum writer member function view-reference-set-type.

[0092] FIG. **82** depicts an informal definition of an exemplary class suitable for class writer operations.

[0093] FIG. **83** depicts an exemplary informal definition of a class writer member function class-entry.

[0094] FIG. **84** depicts an exemplary informal definition of a class writer member function view-member-datum.

[0095] FIG. **85** depicts an informal definition of an exemplary class suitable for module writer operations.

[0096] FIG. **86** depicts an exemplary informal definition of a module writer member function module-entry.

[0097] FIG. **87** depicts an exemplary informal definition of a module writer member function view-class.

[0098] FIG. **88** depicts an exemplary informal definition of a module writer member function view-host-class.

[0099] FIG. **89** depicts an exemplary module specification for an object-oriented module suitable for representation of media files in a file system.

[0100] FIG. **90** depicts an exemplary module definition for an object-oriented module suitable for representation of media files in a file system (1/3).

[0101] FIG. **91** continues the depiction of an exemplary module definition for an object-oriented module suitable for representation of media files in a file system (2/3).

[0102] FIG. **92** concludes the depiction of an exemplary module definition for an object-oriented module suitable for representation of media files in a file system (3/3).

DETAILED DESCRIPTION

1 Terminology

[0103] The present invention concerns the generation of module definitions from module specifications. For the purposes of this invention, a module definition is a collection of interrelated object-oriented class definitions and other elements of an object-oriented computer program. A module specification is an object-oriented data structure typically derived from a textual expression prepared by a human programmer or a computer program. The module specification represents the classes and other elements that will be defined in the module definition.

[0104] Classes may be usefully divided into categorical classes and unitary classes. A categorical class corresponds to a category in a domain-specific model. Thus categorical classes are largely determined by the requirements of the domain to which applications of the specified module are directed. Instances of a categorical class are typically unbounded in number. Categorical classes are usually arranged in a hierarchy or directed acyclic graph reflecting the relations of genera and species among the domain-specific categories.

[0105] Unitary classes relate to the module as a whole. A host is one important unitary class. The host for a module provides a unitary representation of what is typically a multiplicity of objects. The objects are instances of the categorical classes. The host provides access to individual and collected instances according to an organizational scheme which reflects the requirements of the domain. Together with the categorical instances, the host provides a domain-specific object-oriented data structure which is convenient as a target for construction and as a source for processing.

[0106] Another useful unitary class is the viewer. A viewer implements a variation of the well-known visitor pattern to provide convenient access to a domain-specific object-oriented data structure including a host and its categorical instances. The viewer is a collection of handlers, each of which is a member function accepting a categorical instance as its argument. The module definitions include a definition of a base viewer, the behavior of which is determined according to the module specification. An application typically includes one or more specializations of the viewer, derived from the base viewer, with specializations of handlers as necessary to carry out the application tasks.

[0107] The editor is a unitary class similar in structure to the viewer. However, the handlers of a viewer may not modify the data structure, which the editor handlers are permitted make modifications. In visitor contexts corresponding to specialized viewers and editors, dispatch of categorical instances is a useful operation. In dispatching an instance, a virtual member function of the instance is invoked, which in turn invokes a corresponding handler in the visitor. The usefulness arises from the virtuality of the dispatch. The dispatcher may be holding a generic element, but the virtual dispatch will be directed to a specific implementation and hence handler, since specific categorical classes specialize the generic dispatch. This subtle and powerful behavior serves as an organizing principle for computation with visitors. Variations include dispatch to base classes (in which dispatches to the base class handlers is provided), dispatch to merons (i.e. designated meronymic constituents of an instance, described below), and dispatch to aggregates (collections of elements, often heterogeneous relative to a common genus).

[0108] Specializations of the viewer or editor are denoted operants. Operants may use default traversal patterns of their base visitors or specialized traversal to suit particular requirements. Often a combination of default and specialized traversal is most useful. Operants may use member data to conveniently accumulate processing products or to represent important state in an ongoing computation. Operants may delegate to subsidiary operants to partition a computation into convenient segments, with suitable processing state at each level.

[0109] The categorical classes of a module, corresponding to the categories of a domain, are characterized by member data. The form of the member data is specified in the class but each instance has its own copy of member data elements. Member data is characterized by its type. Type is a complex property of data in many object-oriented programming languages. In the present invention, a simplified type system is

used, permitting concise specification and enforcing consistency of usage. Types include value types, reference types, and compound types.

[0110] Value types correspond to scalar data, in which the value of the data accords with the meaning of the data. Scalar data is typically represented by built-in types of a programming language (e.g. int, double, char, in the C language) or by simple classes (e.g. string, date) which may be passed by value (copying elements or structures). Scalar data is used to represent properties of instances, such as identifiers and measured quantities.

[0111] Reference types correspond to references to instances of categories. Reference data is typically represented by pointers or references in a programming language. The value of a pointer is arbitrary and bears no relation to the meaning of the object the pointer represents. Reference data is typically passed by reference (i.e. by copying pointers, not structures). Conceptually, reference data is used to implement assocations among categories in which a first instance is associated with a second instance.

[0112] Compound types correspond to collections. Provision of collections varies widely in programming languages. However, for effective application programming, set, sequence, and map may be sufficient. A set is an unordered collection of elements which does not contain duplicates. A set provides efficient determination of the presence or absence of an element. A set also provide efficient insertion and deletion of an element. Efficient iteration of the constituent elements of a set is another requirement. Efficient in this context means logarithmic in the number of contained elements for determination, insertion, and deletion. Sets may contain reference or value data.

[0113] A sequence is an ordered collection which provides efficient insertion and deletion of elements at either end of the sequence, thus permitting stack, queue, and deque functionality. A sequence may also provide efficient random access to individual elements by offset in the collection order. A sequence also provides efficient iteration of the elements in order. Sequences may contain reference or value data.

[0114] A map provides efficient association between domain and range elements. Given a domain element, a map efficiently produces a corresponding range element or indicates the absence of such an element. Maps must efficiently support insertion and deletion of domain, range pairs. Maps support all combinations of range and domain value and reference. An index denotes a map with a value domain and a reference range. A scale denotes a map with a reference domain and a value range. A binding denotes a map with reference domain and range. A conversion denotes a map with value domain and range.

[0115] Member data having singular or compound reference type may be characterized by a meron qualification. Qualification as a meron indicates that the referenced element should be considered a part of the referencing element. The term is a contraction of the linguistic term "meronym", which is used to distinguish a part in a part/whole relationship. Used in a module specification, meron qualification determines memory management, where applicable, as well as default behavior in the base viewer and the base editor. In the default view of an instance, its merons will be viewed. Non-meron references are ignored by default.

2 Overview

[0116] Refer to FIG. **1**. A generator **1000** processes a synthetic-language specification **1002** to produce generated-code **1004** and generated-documents **1006**. The specification **1002** is processed by a parser **1008**, which constructs an object-oriented data structure encapsulated in a host **1010**. The host **1010** represents the modules specified in the specification **1002**. The constituent module specifications of the host **1010** are processed by a code-generator **1012** to provide the generated-code **1004**. The generated code includes software components written in a conventional object-oriented programming language. The constituent module specifications of the host **1010** are also processed by a document-generator **1014** to provide the generated-documents **1006**. The generated documentation includes descriptions and figures which characterize the modules specified in the specification **1002**.

[0117] The present invention relates to the processing of the constituent module specifications of the host **1010** by the code-generator **1012** to produce the generated-code **1004**. Co-pending applications relate to the language of the specification **1002**, the parser **1008**, and the document-generator **1014**.

[0118] A module specification is represented as an object-oriented data structure, the elements of which are instances of classes representative of domain-specific elements, such as modules, classes, members, types, etc. The domain-specific classes themselves form a module specialized for the representation of modules, hence, a meta-module. The categorical classes and the viewer for the meta-module are described.

[0119] A module specification is processed to produce module definition. The module definition is textual, consisting of programming language code. The processing of the module specification is mediated by several operants, which are specializations of the meta-module viewer. Operants include a module writer, a class writer, a datum writer, and a type definer. These operants are described.

[0120] An example of a module specification and parts of its corresponding module definition is exhibited. The example module provides a representation for files in a hierarchical file system. Finally, alternative embodiments are considered.

[0121] In the exemplary embodiments of the invention exhibited here, many syntactic elements are omitted to enhance clarity and to conform to space limitations. These omissions are intended to highlight the novel and non-obvious aspects of the invention by suppression of unimportant details. The provision of these missing elements would not present any difficulty for one skilled in the art of object-oriented programming.

3 Meta Module Discursive Model

[0122] Refer to FIG. **2**. The meta-module **1016** specifies classes for representation and processing of modules for object-oriented programming. The meta-module **1016** has a viewer class meta-module-viewer **1018**. The meta-module-viewer **1018** provides a viewer base class for the meta-module **1016** with useful defaults and service methods from which specialized view operants may be derived.

[0123] A categorical class entity **1020** represents named elements. A datum id **1022** uniquely identifies a particular entity in a global scope.

[0124] A categorical class module **1024** represents a collection of interrelated classes for object-oriented programming. The module **1024** has genus entity **1020**. A datum classes **1026** represents the collection of classes within a module. The classes **1026** is a meron. A datum host **1028** represents a class, an instance of which encapsulates an application-specific object-oriented data structure corresponding to an instantiation of a module. The host **1028** is a meron. A datum viewer-id **1030** specifies a view operator to be generated; the generated viewer provides a useful base class for read-only operators that process an instantiation of a module. A datum editor-id **1032** specifies an edit operator to be generated; the generated editor provides a useful base class for write-capable operators that process an instantiation of a module.

[0125] A categorical class class **1034** represents a class for object-oriented programming. The class **1034** has genus entity **1020**. A datum genera **1036** represents the collection of classes from which a particular class is derived. A datum members **1038** represents the members of a particular class. The members **1038** is a meron.

[0126] A categorical class host-class **1040** represents an instantiation of a module. The host-class **1040** has genus class **1034**.

[0127] A categorical class operand **1042** represents a typed entity in a scope. The operand **1042** has genus entity **1020**. A datum scope-id **1044** uniquely identifies an operand in a scope. A datum type **1046** specifies the type of an operand.

[0128] A categorical class member **1048** represents a member in a class. The member **1048** has genus operand **1042**.

[0129] A categorical class member-datum **1050** represents a member datum in a class. The member-datum **1050** has genus member **1048**. A datum is-meron **1052** indicates that a member datum is a meronym of the containing instance of the class to which it belongs.

[0130] Refer to FIG. **3**. A categorical class type **1054** characterizes the typing of an operand.

[0131] A categorical class value-type **1056** characterizes a type which is passed by value. The value-type **1056** has genus type **1054**.

[0132] A categorical class bit-type **1058** represents a Boolean value, true or false. The bit-type **1058** has genus value-type **1056**.

[0133] A categorical class text-type **1060** represents a textual value. The text-type **1060** has genus value-type **1056**.

[0134] A categorical class reference-type **1062** characterizes a type which is passed by reference. The reference-type **1062** has genus type **1054**. A datum reference-class-id **1064** identifies the class to which a reference type corresponds.

[0135] A categorical class compound-type **1066** characterizes a type which corresponds to a collection. The compound-type **1066** has genus type **1054**. A datum range **1068** specifies the characteristic subsidiary type of the elements in the compound type.

[0136] A categorical class sequence-type **1070** characterizes a sequence of elements. The sequence-type **1070** has genus compound-type **1066**.

[0137] A categorical class value-sequence-type **1072** characterizes a sequence of value-typed elements. The value-sequence-type **1072** has genus sequence-type **1070**.

[0138] A categorical class reference-sequence-type **1074** characterizes a sequence of reference-typed elements. The reference-sequence-type **1074** has genus sequence-type **1070**.

[0139] A categorical class set-type **1076** characterizes a set of elements. The set-type **1076** has genus compound-type **1066**.

[0140] A categorical class value-set-type **1078** characterizes a set of value-typed elements. The value-set-type **1078** has genus set-type **1076**.

[0141] A categorical class reference-set-type **1080** characterizes a set of reference-typed elements. The reference-set-type **1080** has genus set-type **1076**.

[0142] A categorical class map-type **1082** characterizes a map associating pairs of elements. The map-type **1082** has genus compound-type **1066**. A datum domain **1084** specifies the subsidiary type of the domain elements.

[0143] A categorical class index-map-type **1086** characterizes a map, of which the range elements are of reference type and the domain elements are of value type. The index-map-type **1086** has genus map-type **1082**.

[0144] A categorical class scale-map-type **1088** characterizes a map, of which the range elements are of value type and the domain elements are of reference type. The scale-map-type **1088** has genus map-type **1082**.

[0145] A categorical class bind-map-type **1090** characterizes a map, of which both the range and domain elements are of reference type. The bind-map-type **1090** has genus map-type **1082**.

[0146] A categorical class convert-map-type **1092** characterizes a map, of which both the range and domain elements are of value type. The convert-map-type **1092** has genus map-type **1082**.

3.1 Meta Module Viewer Viewer Class

[0147] Refer to FIG. **4**. The meta-module-viewer **1018** provides a viewer base class for the meta-module **1016** with useful defaults and service methods from which specialized view operants may be derived. A segment class-views **1094** specifies a collection of virtual methods of the meta-module-viewer **1018**, each of which accepts an instance of a particular discursant for read-only processing. The class-views **1094** provides a default implementation which views the merons of the instance and delegates processing to the view(s) of the instance's base class(es). A segment meron-view-dispatches **1096** specifies a collection of methods of the meta-moduleviewer **1018**, each of which accepts an instance of a particular discursant for read-only processing of merons of the instance, if any. A segment base-view-dispatches **1098** specifies a collection of methods of the meta-module-viewer **1018**, each of which accepts an instance of a particular category for read-only processing and delegates processing to view implementations corresponding to the base class(es) of the instance, if any. A segment aggregate-view-dispatches **1100** specifies a collection of methods of the meta-module-viewer **1018**, each of which accepts a collection of instances of a particular category for read-only processing, dispatching processing to each instance of the collection.

[0148] A datum failed **1102** signifies that the viewer has failed. The failed **1102** is initialized to false, indicating that the viewer has not yet failed. The failed **1102** is shown public for ease of exposition; in an alternative embodiment, the failed **1102** could be private, with read access via a const

method, and write access mediated by a method fail **1104**. The fail **1104** encapsulates error reporting and failure indication. The fail **1104** accepts a textual argument which may provide a basis for an error report. The fail **1104** sets the failed **1102**, indicating failure. The fail **1104** returns zero, indicating failure, for convenience in invocation in handlers, which have a boolean return value.

3.2 Categorical Classes

3.2.1 Entity Categorical Class

[0149] Refer to FIG. **5**. The entity **1020** represents named elements. The entity **1020** is a root-level class of the meta-module **1016**. The id **1022** uniquely identifies a particular entity in a global scope. The id **1022** is required. The id **1022** ranges over scalar text. The id **1022** has singleton arity. A method dispatch-view **1106** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the entity **1020**.

[0150] Refer to FIG. **6**, which depicts a segment of the meta-module-viewer **1018** corresponding to the entity **1020**. A method view **1108** provides a default implementation for an operant handler processing instances of the entity **1020**. The view **1108** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1110**. Processing by base handlers is delegated to a method view-base **1112**. The view **1108** is a constituent of the class-views **1094**. The view-base **1112** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the entity **1020**. The view-base **1112** is a constituent of the base-view-dispatches **1098**. The view-merons **1110** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the entity **1020**. The entity **1020** lacks any merons, so the view-merons **1110** is trivially successful. The view-merons **1110** is a constituent of the meron-view-dispatches **1096**.

3.2.2 Module Categorical Class

[0151] Refer to FIG. **7**. The module **1024** represents a collection of interrelated classes for object-oriented programming. The module **1024** has genus entity **1020**. The classes **1026** represents the collection of classes within a module. The classes **1026** ranges over instances of the class class **1034**. The classes **1026** is a meron. The classes **1026** has sequence arity. The host **1028** represents a class, an instance of which encapsulates an application-specific object-oriented data structure corresponding to an instantiation of a module. The host **1028** is the target for module-scoped member specifications. The host **1028** instance may be denoted as a model object or a document object. The host **1028** may contain resolver, depository, factory, reflector, etc. according to specification. The host **1028** ranges over instances of the class host-class **1040**. The host **1028** is a meron. The host **1028** has singleton arity. The viewer-id **1030** specifies a view operator to be generated; the generated viewer provides a useful base class for read-only operators that process an instantiation of a module. The viewer-id **1030** ranges over scalar text. The viewer-id **1030** has singleton arity. The editor-id **1032** specifies an edit operator to be generated; the generated editor provides a useful base class for write-capable operators that process an instantiation of a module. The editor-id **1032** ranges over scalar text. The editor-id **1032** has singleton arity. A method dispatch-view **1114** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the module **1024**.

[0152] Refer to FIG. **8**, which depicts a segment of the meta-module-viewer **1018** corresponding to the module **1024**. A method view **1116** provides a default implementation for an operant handler processing instances of the module **1024**. The view **1116** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1118**. Processing by base handlers is delegated to a method view-base **1120**. The view **1116** is a constituent of the class-views **1094**. The view-base **1120** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the module **1024**. The view-base **1120** is a constituent of the base-view-dispatches **1098**. The view-merons **1118** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the module **1024**. The view **1116** dispatches an aggregate view to the classes **1026**, a sequence of instances of the class **1034**, making use of a method view-class-sequence **1122**. The view-merons **1118** is a constituent of the meron-view-dispatches **1096**. The view-class-sequence **1122** dispatches a view to each constituent of the supplied sequence of instances of the class **1034**. The view-class-sequence **1122** is supplied with a sequence of instances of the class **1034**. The view dispatch to each constituent instance of the class **1034** makes use of the dispatch-view **1124**. The view-class-sequence **1122** is a constituent of the aggregate-view-dispatches **1100**.

3.2.3 Class Categorical Class

[0153] Refer to FIG. **9**. The class **1034** represents a class for object-oriented programming. The class **1034** has genus entity **1020**. The genera **1036** represents the collection of classes from which a particular class is derived. The genera **1036** includes the closest generalizations of a particular class. The genera **1036** ranges over instances of the class class **1034**. The genera **1036** has sequence arity. The members **1038** represents the members of a particular class. The members **1038** may include data members and function members. The members **1038** ranges over instances of the class member **1048**. The members **1038** is a meron. The members **1038** has sequence arity. A method dispatch-view **1124** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the class **1034**.

[0154] Refer to FIG., **10**, which depicts a segment of the meta-module-viewer **1018** corresponding to the class **1034**. A method view **1126** provides a default implementation for an operant handler processing instances of the class **1034**. The view **1126** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1128**. Processing by base handlers is delegated to a method view-base **1130**. The view **1126** is a constituent of the class-views **1094**. The view-base **1130** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the class **1034**. The view-base **1130** is a constituent of the base-view-dispatches **1098**. The view-merons **1128** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the class **1034**. The view **1126** dispatches an aggregate view to the members **1038**, a sequence of instances of the member **1048**,

making use of a method view-member-sequence **1132**. The view-merons **1128** is a constituent of the meron-view-dispatches **1096**. The view-member-sequence **1132** dispatches a view to each constituent of the supplied sequence of instances of the member **1048**. The view-member-sequence **1132** is supplied with a sequence of instances of the member **1048**. The view dispatch to each constituent instance of the member **1048** makes use of the dispatch-view **1134**. The view-member-sequence **1132** is a constituent of the aggregate-view-dispatches **1100**.

3.2.4 Host Class Categorical Class

[0155] Refer to FIG. **11**. The host-class **1040** represents an instantiation of a module. The host-class **1040** represents the totality of an application-specific object-oriented data structure. The host-class **1040** corresponds to the host **1028**. The host-class **1040** typically occurs in a singleton instance per application. The host-class **1040** provides a useful target for parsing, markup, etc. The host-class **1040** has genus class **1034**. A method dispatch-view **1136** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the host-class **1040**.

[0156] Refer to FIG. **12**, which depicts a segment of the meta-module-viewer **1018** corresponding to the host-class **1040**. A method view **1138** provides a default implementation for an operant handler processing instances of the host-class **1040**. The view **1138** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1140**. Processing by base handlers is delegated to a method view-base **1142**. The view **1138** is a constituent of the class-views **1094**. The view-base **1142** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the host-class **1040**. The view-base **1142** is a constituent of the base-view-dispatches **1098**. The view-merons **1140** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the host-class **1040**. The host-class **1040** lacks any merons, so the view-merons **1140** is trivially successful. The view-merons **1140** is a constituent of the meron-view-dispatches **1096**.

3.2.5 Operand Categorical Class

[0157] Refer to FIG. **13**. The operand **1042** represents a typed entity in a scope. The operand **1042** has genus entity **1020**. The scope-id **1044** uniquely identifies an operand in a scope. The scope-id **1044** is applicable to members in a class scope and arguments in a member function scope. The scope-id **1044** ranges over scalar text. The scope-id **1044** has singleton arity. The type **1046** specifies the type of an operand. The type **1046** ranges over instances of the class type **1054**. The type **1046** has singleton arity. A method dispatch-view **1144** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the operand **1042**.

[0158] Refer to FIG. **14**, which depicts a segment of the meta-module-viewer **1018** corresponding to the operand **1042**. A method view **1146** provides a default implementation for an operant handler processing instances of the operand **1042**. The view **1146** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1148**. Processing by base handlers is delegated to a method view-base **1150**. The view **1146** is a constituent of the class-views **1094**. The view-base **1150** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the operand **1042**. The view-base **1150** is a constituent of the base-view-dispatches **1098**. The view-merons **1148** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the operand **1042**. The operand **1042** lacks any merons, so the view-merons **1148** is trivially successful. The view-merons **1148** is a constituent of the meron-view-dispatches **1096**.

3.2.6 Member Categorical Class

[0159] Refer to FIG. **15**. The member **1048** represents a member in a class. The member **1048** has genus operand **1042**. A method dispatch-view **1134** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the member **1048**.

[0160] Refer to FIG. **16**, which depicts a segment of the meta-module-viewer **1018** corresponding to the member **1048**. A method view **1152** provides a default implementation for an operant handler processing instances of the member **1048**. The view **1152** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1154**. Processing by base handlers is delegated to a method view-base **1156**. The view **1152** is a constituent of the class-views **1094**. The view-base **1156** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the member **1048**. The view-base **1156** is a constituent of the base-view-dispatches **1098**. The view-merons **1154** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the member **1048**. The member **1048** lacks any merons, so the view-merons **1154** is trivially successful. The view-merons **1154** is a constituent of the meron-view-dispatches **1096**.

3.2.7 Member Datum Categorical Class

[0161] Refer to FIG. **17**. The member-datum **1050** represents a member datum in a class. The member-datum **1050** has genus member **1048**. The is-meron **1052** indicates that a member datum is a meronym of the containing instance of the class to which it belongs. The is-meron **1052** indicates that the member datum is to be deleted on deletion of the containing instance. The is-meron **1052** indicates that, by default, viewers and editors of the module should recursively visit the member datum when visiting the containing instance. The is-meron **1052** ranges over scalar boolean. The is-meron **1052** has singleton arity. A method dispatch-view **1158** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the member-datum **1050**.

[0162] Refer to FIG. **18**, which depicts a segment of the meta-module-viewer **1018** corresponding to the member-datum **1050**. A method view **1160** provides a default implementation for an operant handler processing instances of the member-datum **1050**. The view **1160** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1162**. Processing by base handlers is delegated to a method view-base **1164**. The view **1160** is a constituent of the class-views **1094**. The view-base **1164** delegates view dis-

patch to the base classes, if any, for an operant handler processing instances of the member-datum **1050**. The view-base **1164** is a constituent of the base-view-dispatches **1098**. The view-merons **1162** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the member-datum **1050**. The member-datum **1050** lacks any merons, so the view-merons **1162** is trivially successful. The view-merons **1162** is a constituent of the meron-view-dispatches **1096**.

3.2.8 Type Categorical Class

[0163] Refer to FIG. **19**. The type **1054** characterizes the typing of an operand. The type **1054** is a root-level class of the meta-module **1016**. A method dispatch-view **1166** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the type **1054**.

[0164] Refer to FIG. **20**, which depicts a segment of the meta-module-viewer **1018** corresponding to the type **1054**. A method view **1168** provides a default implementation for an operant handler processing instances of the type **1054**. The view **1168** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1170**. Processing by base handlers is delegated to a method view-base **1172**. The view **1168** is a constituent of the class-views **1094**. The view-base **1172** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the type **1054**. The view-base **1172** is a constituent of the base-view-dispatches **1098**. The view-merons **1170** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the type **1054**. The type **1054** lacks any merons, so the view-merons **1170** is trivially successful. The view-merons **1170** is a constituent of the meron-view-dispatches **1096**.

3.2.9 Value Type Categorical Class

[0165] Refer to FIG. **21**. The value-type **1056** characterizes a type which is passed by value. The value-type **1056** has genus type **1054**. A method dispatch-view **1174** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the value-type **1056**.

[0166] Refer to FIG. **22**, which depicts a segment of the meta-module-viewer **1018** corresponding to the value-type **1056**. A method view **1176** provides a default implementation for an operant handler processing instances of the value-type **1056**. The view **1176** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1178**. Processing by base handlers is delegated to a method view-base **1180**. The view **1176** is a constituent of the class-views **1094**. The view-base **1180** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the value-type **1056**. The view-base **1180** is a constituent of the base-view-dispatches **1098**. The view-merons **1178** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the value-type **1056**. The value-type **1056** lacks any merons, so the view-merons **1178** is trivially successful. The view-merons **1178** is a constituent of the meron-view-dispatches **1096**.

3.2.10 Bit Type Categorical Class

[0167] Refer to FIG. **23**. The bit-type **1058** represents a Boolean value, true or false. The bit-type **1058** has genus value-type **1056**. A method dispatch-view **1182** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the bit-type **1058**.

[0168] Refer to FIG. **24**, which depicts a segment of the meta-module-viewer **1018** corresponding to the bit-type **1058**. A method view **1184** provides a default implementation for an operant handler processing instances of the bit-type **1058**. The view **1184** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1186**. Processing by base handlers is delegated to a method view-base **1188**. The view **1184** is a constituent of the class-views **1094**. The view-base **1188** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the bit-type **1058**. The view-base **1188** is a constituent of the base-view-dispatches **1098**. The view-merons **1186** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the bit-type **1058**. The bit-type **1058** lacks any merons, so the view-merons **1186** is trivially successful. The view-merons **1186** is a constituent of the meron-view-dispatches **1096**.

3.2.11 Text Type Categorical Class

[0169] Refer to FIG. **25**. The text-type **1060** represents a textual value. The text-type **1060** has genus value-type **1056**. A method dispatch-view **1190** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the text-type **1060**.

[0170] Refer to FIG. **26**, which depicts a segment of the meta-module-viewer **1018** corresponding to the text-type **1060**. A method view **1192** provides a default implementation for an operant handler processing instances of the text-type **1060**. The view **1192** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1194**. Processing by base handlers is delegated to a method view-base **1196**. The view **1192** is a constituent of the class-views **1094**. The view-base **1196** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the text-type **1060**. The view-base **1196** is a constituent of the base-view-dispatches **1098**. The view-merons **1194** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the text-type **1060**. The text-type **1060** lacks any merons, so the view-merons **1194** is trivially successful. The view-merons **1194** is a constituent of the meron-view-dispatches **1096**.

3.2.12 Reference Type Categorical Class

[0171] Refer to FIG. **27**. The reference-type **1062** characterizes a type which is passed by reference. The reference-type **1062** corresponds to a class. The reference-type **1062** has genus type **1054**. The reference-class-id **1064** identifies the class to which a reference type corresponds. The refer-

ence-class-id **1064** ranges over scalar text. The reference-class-id **1064** has singleton arity. A method dispatch-view **1198** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the reference-type **1062**.

[0172] Refer to FIG. **28**, which depicts a segment of the meta-module-viewer **1018** corresponding to the reference-type **1062**. A method view **1200** provides a default implementation for an operant handler processing instances of the reference-type **1062**. The view **1200** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1202**. Processing by base handlers is delegated to a method view-base **1.204**. The view **1200** is a constituent of the class-views **1094**. The view-base **1204** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the reference-type **1062**. The view-base **1204** is a constituent of the base-view-dispatches **1098**. The view-merons **1202** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the reference-type **1062**. The reference-type **1062** lacks any merons, so the view-merons **1202** is trivially successful. The view-merons **1202** is a constituent of the meron-view-dispatches **1096**.

3.2.13 Compound Type Categorical Class

[0173] Refer to FIG. **29**. The compound-type **1066** characterizes a type which corresponds to a collection. The compound-type **1066** is parameterized by one or more subsidiary types. The compound-type **1066** has genus type **1054**. The range **1068** specifies the characteristic subsidiary type of the elements in the compound type. The range **1068** ranges over instances of the class type **1054**. The range **1068** has singleton arity. A method dispatch-view **1206** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the compound-type **1066**.

[0174] Refer to FIG. **30**, which depicts a segment of the meta-module-viewer **1018** corresponding to the compound-type **1066**. A method view **1208** provides a default implementation for an operant handler processing instances of the compound-type **1066**. The view **1208** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1210**. Processing by base handlers is delegated to a method view-base **1212**. The view **1208** is a constituent of the class-views **1094**. The view-base **1212** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the compound-type **1066**. The view-base **1212** is a constituent of the base-view-dispatches **1098**. The view-merons **1210** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the compound-type **1066**. The compound-type **1066** lacks any merons, so the view-merons **1210** is trivially successful. The view-merons **1210** is a constituent of the meron-view-dispatches **1096**.

3.2.14 Sequence Type Categorical Class

[0175] Refer to FIG. **31**. The sequence-type **1070** characterizes a sequence of elements. The sequence-type **1070** specifies a compound element that permits efficient addition or removal of elements at the front or back of the sequence. The sequence-type **1070** specifies a compound element that permits direct access to elements by position in the sequence. The sequence-type **1070** specifies a compound element that permits iteration of the elements in the sequence. The sequence-type **1070** has genus compound-type **1066**. A method dispatch-view **1214** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the sequence-type **1070**.

[0176] Refer to FIG. **32**, which depicts a segment of the meta-module-viewer **1018** corresponding to the sequence-type **1070**. A method view **1216** provides a default implementation for an operant handler processing instances of the sequence-type **1070**. The view **1216** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1218**. Processing by base handlers is delegated to a method view-base **1220**. The view **1216** is a constituent of the class-views **1094**. The view-base **1220** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the sequence-type **1070**. The view-base **1220** is a constituent of the base-view-dispatches **1098**. The view-merons **1218** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the sequence-type **1070**. The sequence-type **1070** lacks any merons, so the view-merons **1218** is trivially successful. The view-merons **1218** is a constituent of the meron-view-dispatches **1096**.

3.2.15 Value Sequence Type Categorical Class

[0177] Refer to FIG. **33**. The value-sequence-type **1072** characterizes a sequence of value-typed elements. The value-sequence-type **1072** has genus sequence-type **1070**. A method dispatch-view **1222** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the value-sequence-type **1072**.

[0178] Refer to FIG. **34**, which depicts a segment of the meta-module-viewer **1018** corresponding to the value-sequence-type **1072**. A method view **1224** provides a default implementation for an operant handler processing instances of the value-sequence-type **1072**. The view **1224** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1226**. Processing by base handlers is delegated to a method view-base **1228**. The view **1224** is a constituent of the class-views **1094**. The view-base **1228** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the value-sequence-type **1072**. The view-base **1228** is a constituent of the base-view-dispatches **1098**. The view-merons **1226** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the value-sequence-type **1072**. The value-sequence-type **1072** lacks any merons, so the view-merons **1226** is trivially successful. The view-merons **1226** is a constituent of the meron-view-dispatches **1096**.

3.2.16 Reference Sequence Type Categorical Class

[0179] Refer to FIG. **35**. The reference-sequence-type **1074** characterizes a sequence of reference-typed elements. The reference-sequence-type **1074** has genus sequence-type **1070**. A method dispatch-view **1230** specifies a virtual

member function which specializes a generic view dispatch to the specific view corresponding to the reference-sequence-type **1074**.

[0180] Refer to FIG. **36**, which depicts a segment of the meta-module-viewer **1018** corresponding to the reference-sequence-type **1074**. A method view **1232** provides a default implementation for an operant handler processing instances of the reference-sequence-type **1074**. The view **1232** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1234**. Processing by base handlers is delegated to a method view-base **1236**. The view **1232** is a constituent of the class-views **1094**. The view-base **1236** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the reference-sequence-type **1074**. The view-base **1236** is a constituent of the base-view-dispatches **1098**. The view-merons **1234** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the reference-sequence-type **1074**. The reference-sequence-type **1074** lacks any merons, so the view-merons **1234** is trivially successful. The view-merons **1234** is a constituent of the meron-view-dispatches **1096**.

3.2.17 Set Type Categorical Class

[0181] Refer to FIG. **37**. The set-type **1076** characterizes a set of elements. The set-type **1076** specifies a compound element that permits efficient determination of the presence or absence of a particular element in the set. The set-type **1076** specifies a compound element that permits iteration of elements in the set. The set-type **1076** has genus compound-type **1066**. A method dispatch-view **1238** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the set-type **1076**.

[0182] Refer to FIG. **38**, which depicts a segment of the meta-module-viewer **1018** corresponding to the set-type **1076**. A method view **1240** provides a default implementation for an operant handler processing instances of the set-type **1076**. The view **1240** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1242**. Processing by base handlers is delegated to a method view-base **1244**. The view **1240** is a constituent of the class-views **1094**. The view-base **1244** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the set-type **1076**. The view-base **1244** is a constituent of the base-view-dispatches **1098**. The view-merons **1242** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the set-type **1076**. The set-type **1076** lacks any merons, so the view-merons **1242** is trivially successful. The view-merons **1242** is a constituent of the meron-view-dispatches **1096**.

3.2.18 Value Set Type Categorical Class

[0183] Refer to FIG. **39**. The value-set-type **1078** characterizes a set of value-typed elements. The value-set-type **1078** has genus set-type **1076**. A method dispatch-view **1246** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the value-set-type **1078**.

[0184] Refer to FIG. **40**, which depicts a segment of the meta-module-viewer **1018** corresponding to the value-set-type **1078**. A method view **1248** provides a default implementation for an operant handler processing instances of the value-set-type **1078**. The view **1248** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1250**. Processing by base handlers is delegated to a method view-base **1252**. The view **1248** is a constituent of the class-views **1094**. The view-base **1252** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the value-set-type **1078**. The view-base **1252** is a constituent of the base-view-dispatches **1098**. The view-merons **1250** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the value-set-type **1078**. The value-set-type **1078** lacks any merons, so the view-merons **1250** is trivially successful. The view-merons **1250** is a constituent of the meron-view-dispatches **1096**.

3.2.19 Reference Set Type Categorical Class

[0185] Refer to FIG. **41**. The reference-set-type **1080** characterizes a set of reference-typed elements. The reference-set-type **1080** has genus set-type **1076**. A method dispatch-view **1254** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the reference-set-type **1080**.

[0186] Refer to FIG. **42**, which depicts a segment of the meta-module-viewer **1018** corresponding to the reference-set-type **1080**. A method view **1256** provides a default implementation for an operant handler processing instances of the reference-set-type **1080**. The view **1256** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1258**. Processing by base handlers is delegated to a method view-base **1260**. The view **1256** is a constituent of the class-views **1094**. The view-base **1260** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the reference-set-type **1080**. The view-base **1260** is a constituent of the base-view-dispatches **1098**. The view-merons **1258** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the reference-set-type **1080**. The reference-set-type **1080** lacks any merons, so the view-merons **1258** is trivially successful. The view-merons **1258** is a constituent of the meron-view-dispatches **1096**.

3.2.20 Map Type Categorical Class

[0187] Refer to FIG. **43**. The map-type **1082** characterizes a map associating pairs of elements. The map-type **1082** specifies a compound element that permits association of a range element with a supplied domain element. The map-type **1082** specifies a compound element that permits iteration of pairs. The map-type **1082** has genus compound-type **1066**. The domain **1084** specifies the subsidiary type of the domain elements. The domain **1084** ranges over instances of the class type **1054**. The domain **1084** has singleton arity. A method dispatch-view **1262** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the map-type **1082**.

[0188] Refer to FIG. **44**, which depicts a segment of the meta-module-viewer **1018** corresponding to the map-type **1082**. A method view **1264** provides a default implementation for an operant handler processing instances of the map-type **1082**. The view **1264** delegates processing to

merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1266**. Processing by base handlers is delegated to a method view-base **1268**. The view **1264** is a constituent of the class-views **1094**. The view-base **1268** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the map-type **1082**. The view-base **1268** is a constituent of the base-view-dispatches **1098**. The view-merons **1266** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the map-type **1082**. The map-type **1082** lacks any merons, so the view-merons **1266** is trivially successful. The view-merons **1266** is a constituent of the meron-view-dispatches **1096**.

3.2.21 Index Map Type Categorical Class

[0189] Refer to FIG. **45**. The index-map-type **1086** characterizes a map, of which the range elements are of reference type and the domain elements are of value type. The index-map-type **1086** has genus map-type **1082**. A method dispatch-view **1270** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the index-map-type **1086**.

[0190] Refer to FIG. **46**, which depicts a segment of the meta-module-viewer **1018** corresponding to the index-map-type **1086**. A method view **1272** provides a default implementation for an operant handler processing instances of the index-map-type **1086**. The view **1272** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1274**. Processing by base handlers is delegated to a method view-base **1276**. The view **1272** is a constituent of the class-views **1094**. The view-base **1276** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the index-map-type **1086**. The view-base **1276** is a constituent of the base-view-dispatches **1098**. The view-merons **1274** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the index-map-type **1086**. The index-map-type **1086** lacks any merons, so the view-merons **1274** is trivially successful. The view-merons **1274** is a constituent of the meron-view-dispatches **1096**.

3.2.22 Scale Map Type Categorical Class

[0191] Refer to FIG. **47**. The scale-map-type **1088** characterizes a map, of which the range elements are of value type and the domain elements are of reference type. The scale-map-type **1088** has genus map-type **1082**. A method dispatch-view **1278** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the scale-map-type **1088**.

[0192] Refer to FIG. **48**, which depicts a segment of the meta-module-viewer **1018** corresponding to the scale-map-type **1088**. A method view **1280** provides a default implementation for an operant handler processing instances of the scale-map-type **1088**. The view **1280** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1282**. Processing by base handlers is delegated to a method view-base **1284**. The view **1280** is a constituent of the class-views **1094**. The view-base **1284** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the scale-map-type **1088**. The view-base **1284** is a constituent of the base-view-dispatches **1098**. The view-merons **1282** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the scale-map-type **1088**. The scale-map-type **1088** lacks any merons, so the view-merons **1282** is trivially successful. The view-merons **1282** is a constituent of the meron-view-dispatches **1096**.

3.2.23 Bind Map Type Categorical Class

[0193] Refer to FIG. **49**. The bind-map-type **1090** characterizes a map, of which both the range and domain elements are of reference type. The bind-map-type **1090** has genus map-type **1082**. A method dispatch-view **1286** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the bind-map-type **1090**.

[0194] Refer to FIG. **50**, which depicts a segment of the meta-module-viewer **1018** corresponding to the bind-map-type **1090**. A method view **1288** provides a default implementation for an operant handler processing instances of the bind-map-type **1090**. The view **1288** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1290**. Processing by base handlers is delegated to a method view-base **1292**. The view **1288** is a constituent of the class-views **1094**. The view-base **1292** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the bind-map-type **1090**. The view-base **1292** is a constituent of the base-view-dispatches **1098**. The view-merons **1290** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the bind-map-type **1090**. The bind-map-type **1090** lacks any merons, so the view-merons **1290** is trivially successful. The view-merons **1290** is a constituent of the meron-view-dispatches **1096**.

3.2.24 Convert Map Type Categorical Class

[0195] Refer to FIG. **51**. The convert-map-type **1092** characterizes a map, of which both the range and domain elements are of value type. The convert-map-type **1092** has genus map-type **1082**. A method dispatch-view **1294** specifies a virtual member function which specializes a generic view dispatch to the specific view corresponding to the convert-map-type **1092**.

[0196] Refer to FIG. **52**, which depicts a segment of the meta-module-viewer **1018** corresponding to the convert-map-type **1092**. A method view **1296** provides a default implementation for an operant handler processing instances of the convert-map-type **1092**. The view **1296** delegates processing to merons and view handlers for base classes, if any. Processing of merons is delegated to a method view-merons **1298**. Processing by base handlers is delegated to a method view-base **1300**. The view **1296** is a constituent of the class-views **1094**. The view-base **1300** delegates view dispatch to the base classes, if any, for an operant handler processing instances of the convert-map-type **1092**. The view-base **1300** is a constituent of the base-view-dispatches **1098**. The view-merons **1298** delegates view dispatches to the constituent meron members, if any, for an operant handler processing instances of the convert-map-type **1092**. The convert-map-type **1092** lacks any merons, so the view-

merons **1298** is trivially successful. The view-merons **1298** is a constituent of the meron-view-dispatches **1096**.

4 Operant Specifications

4.1 Type Definer View Operant

[0197] Refer to FIG. **53**. A view-operant type definer **1302** processes an instance of the type **1054**.

[0198] The type definer **1302** has base the meta-module-viewer **1018**. A datum type identifier **1304** defines an identifier which corresponds to the specified type. A datum type definition **1306** provides a programming-language definition for the specified type. A datum typedef definition **1308** defines a particular typedef, corresponding to the compound-type **1066**. A datum include requirement **1310** defines a required include file for the type, if any. A datum template specifier **1312** defines a template identifier for the type, if appropriate.

[0199] Refer to FIG. **54**. A view-entry type-entry **1314** receives an instance of the type **1054**. A datum type **1316** provides an argument containing the instance of the type **1054** upon which the type-entry **1314** operates. A step **1318** dispatches the type **1316**.

[0200] Refer to FIG. **55**. A view-handler view-bit-type **1320** receives an instance of the bit-type **1058**, and operates on the type identifier **1304**. A datum bit-type **1322** provides an argument containing the instance of the bit-type **1058** upon which the view-bit-type **1320** operates. A step **1324** writes a boolean specifier to the type identifier **1304**. A step **1326** returns boolean true, indicating success in processing the supplied bit-type **1322**.

[0201] Refer to FIG. **56**. A view-handler view-text-type **1328** receives an instance of the text-type **1060**, and operates on the type identifier **1304**. A datum text-type **1330** provides an argument containing the instance of the text-type **1060** upon which the view-text-type **1328** operates. A step **1332** writes a text specifier to the type identifier **1304**. A step **1334** returns boolean true, indicating success in processing the supplied text-type **1330**.

[0202] Refer to FIG. **57**. A view-handler view-reference-type **1336** receives an instance of the reference-type **1062**, and operates on the type identifier **1304** and the type definition **1306**. A datum reference-type **1338** provides an argument containing the instance of the reference-type **1062** upon which the view-reference-type **1336** operates. A step **1340** writes the type identifier **1304** using the reference-class-id **1064** of the reference-type **1338**. A step **1342** writes a pointer specifier to the type definition **1306** using the reference-class-id **1064** of the reference-type **1338**. A step **1344** returns boolean true, indicating success in processing the supplied reference-type **1338**.

[0203] Refer to FIG. **58**. A view-handler view-compound-type **1346** receives an instance of the compound-type **1066**, and operates on the typedef definition **1308**, the type identifier **1304**, and the type definition **1306**. A datum compound-type **1348** provides an argument containing the instance of the compound-type **1066** upon which the view-compound-type **1346** operates. A step **1350** writes the typedef definition **1308** using the type identifier **1304** and the type definition **1306**. A step **1352** returns boolean true, indicating success in processing the supplied compound-type **1348**.

[0204] Refer to FIG. **59**. A view-handler view-sequence-type **1354** receives an instance of the sequence-type **1070**, and operates on the type identifier **1304**, the type definition **1306**, the include requirement **1310**, and the template specifier **1312**. A datum sequence-type **1356** provides an argument containing the instance of the sequence-type **1070** upon which the view-sequence-type **1354** operates. A step **1358** writes a sequence include directive to the include requirement **1310**. A step **1360** delegates to the range **1068** of the sequence-type **1356**. A step **1362** writes the type definition **1306** using the type identifier **1304** of the delegate and the template specifier **1312**. A step **1364** writes the type identifier **1304** using the type identifier **1304** of the delegate. A step **1366** dispatches to the bases of the sequence-type **1356**. A step **1368** returns boolean true, indicating success in processing the supplied sequence-type **1356**.

[0205] Refer to FIG. **60**. A view-handler view-reference-sequence-type **1370** receives an instance of the reference-sequence-type **1074**, and operates on the template specifier **1312**. A datum reference-sequence-type **1372** provides an argument containing the instance of the reference-sequence-type **1074** upon which the view-reference-sequence-type **1370** operates. A step **1374** writes a pointer sequence template specifier to the template specifier **1312**. A step **1376** dispatches to the bases of the reference-sequence-type **1372**. A step **1378** returns boolean true, indicating success in processing the supplied reference-sequence-type **1372**.

[0206] Refer to FIG. **61**. A view-handler view-value-sequence-type **1380** receives an instance of the value-sequence-type **1072**, and operates on the template specifier **1312**. A datum value-sequence-type **1382** provides an argument containing the instance of the value-sequence-type **1072** upon which the view-value-sequence-type **1380** operates. A step **1384** writes a sequence template specifier to the template specifier **1312**. A step **1386** dispatches to the bases of the value-sequence-type **1382**. A step **1388** returns boolean true, indicating success in processing the supplied value-sequence-type **1382**.

[0207] Refer to FIG. **62**. A view-handler view-set-type **1390** receives an instance of the set-type **1076**, and operates on the type identifier **1304**, the type definition **1306**, the include requirement **1310**, and the template specifier **1312**. A datum set-type **1392** provides an argument containing the instance of the set-type **1076** upon which the view-set-type **1390** operates. A step **1394** writes a set include directive to the include requirement **1310**. A step **1396** delegates to the range **1068** of the set-type **1392**. A step **1398** writes the type definition **1306** using the type identifier **1304** of the delegate and the template specifier **1312**. A step **1400** writes the type identifier **1304** using the type identifier **1304** of the delegate. A step **1402** dispatches to the bases of the set-type **1392**. A step **1404** returns boolean true, indicating success in processing the supplied set-type **1392**.

[0208] Refer to FIG. **63**. A view-handler view-reference-set-type **1406** receives an instance of the reference-set-type **1080**, and operates on the template specifier **1312**. A datum reference-set-type **1408** provides an argument containing the instance of the reference-set-type **1080** upon which the view-reference-set-type **1406** operates. A step **1410** writes a pointer set template specifier to the template specifier **1312**. A step **1412** dispatches to the bases of the reference-set-type **1408**. A step **1414** returns boolean true, indicating success in processing the supplied reference-set-type **1408**.

[0209] Refer to FIG. **64**. A view-handler view-value-set-type **1416** receives an instance of the value-set-type **1078**, and operates on the template specifier **1312**. A datum value-

set-type **1418** provides an argument containing the instance of the value-set-type **1078** upon which the view-value-set-type **1416** operates. A step **1420** writes a set template specifier to the template specifier **1312**. A step **1422** dispatches to the bases of the value-set-type **1418**. A step **1424** returns boolean true, indicating success in processing the supplied value-set-type **1418**.

[0210] Refer to FIG. **65**. A view-handler view-map-type **1426** receives an instance of the map-type **1082**, and operates on the type identifier **1304**, the type definition **1306**, the include requirement **1310**, and the template specifier **1312**. A datum map-type **1428** provides an argument containing the instance of the map-type **1082** upon which the view-map-type **1426** operates. A step **1430** writes a map include directive to the include requirement **1310**. A step **1432** delegates to the range **1068** of the map-type **1428**. A step **1434** delegates to the domain **1084** of the map-type **1428**. A step **1436** writes the type definition **1306** using the type identifier **1304** of the delegate, the type identifier **1304** of the delegate, and the template specifier **1312**. A step **1438** writes the type identifier **1304** using the type identifier **1304** of the delegate, the type identifier **1304** of the delegate, and the template specifier **1312**. A step **1440** dispatches to the bases of the map-type **1428**. A step **1442** returns boolean true, indicating success in processing the supplied map-type **1428**.

[0211] Refer to FIG. **66**. A view-handler view-index-map-type **1444** receives an instance of the index-map-type **1086**, and operates on the template specifier **1312**. A datum index-map-type **1446** provides an argument containing the instance of the index-map-type **1086** upon which the view-index-map-type **1444** operates. A step **1448** writes an index template specifier to the template specifier **1312**. A step **1450** dispatches to the bases of the index-map-type **1446**. A step **1452** returns boolean true, indicating success in processing the supplied index-map-type **1446**.

[0212] Refer to FIG. **67**. A view-handler view-scale-map-type **1454** receives an instance of the scale-map-type **1088**, and operates on the template specifier **1312**. A datum scale-map-type **1456** provides an argument containing the instance of the scale-map-type **1088** upon which the view-scale-map-type **1454** operates. A step **1458** writes a scale template specifier to the template specifier **1312**. A step **1460** dispatches to the bases of the scale-map-type **1456**. A step **1462** returns boolean true, indicating success in processing the supplied scale-map-type **1456**.

[0213] Refer to FIG. **68**. A view-handler view-bind-map-type **1464** receives an instance of the bind-map-type **1090**, and operates on the template specifier **1312**. A datum bind-map-type **1466** provides an argument containing the instance of the bind-map-type **1090** upon which the view-bind-map-type **1464** operates. A step **1468** writes a bind template specifier to the template specifier **1312**. A step **1470** dispatches to the bases of the bind-map-type **1466**. A step **1472** returns boolean true, indicating success in processing the supplied bind-map-type **1466**.

[0214] Refer to FIG. **69**. A view-handler view-convert-map-type **1474** receives an instance of the convert-map-type **1092**, and operates on the template specifier **1312**. A datum convert-map-type **1476** provides an argument containing the instance of the convert-map-type **1092** upon which the view-convert-map-type **1474** operates. A step **1478** writes a convert template specifier to the template specifier **1312**. A step **1480** dispatches to the bases of the convert-map-type **1476**. A step **1482** returns boolean true, indicating success in processing the supplied convert-map-type **1476**.

4.2 Forwards Writer View Operant

[0215] Refer to FIG. **70**. A view-operant forwards writer **1484** processes an instance of the module **1024**. The forwards writer **1484** has base the meta-module-viewer **1018**. A datum viewer forward declaration **1486** declares a viewer class. A datum forward class declarations **1488** defines forward class declarations for constituents of the specified module. A datum editor forward declaration **1490** declares an editor class. A datum categorical class forward declaration **1492** declares a categorical class. A datum include directives **1494** defines directives for include files required by the specified module. A datum typedef definitions **1496** associates type identifiers with type definitions corresponding to compound types used by the specified module. A datum host forward declaration **1498** declares a host class.

[0216] Refer to FIG. **71**. A view-entry module-entry **1500** receives an instance of the module **1024**, and operates on the forward class declarations **1488**, the viewer forward declaration **1486**, and the editor forward declaration **1490**. A datum module **1502** provides an argument containing the instance of the module **1024** upon which the module-entry **1500** operates. A step **1504** dispatches the classes **1026** of the module **1502**. A step **1506** dispatches the host **1028** of the module **1502**. A step **1508** writes the viewer forward declaration **1486** using the viewer-id **1030** of the module **1502**. A step **1510** accumulates the viewer forward declaration **1486** to the forward class declarations **1488**. A step **1512** writes the editor forward declaration **1490** using the editor-id **1032** of the module **1502**. A step **1514** accumulates the editor forward declaration **1490** to the forward class declarations **1488**.

[0217] Refer to FIG. **72**. A view-handler view class **1516** receives an instance of the class **1034**, and operates on the forward class declarations **1488** and the categorical class forward declaration **1492**. A datum class **1518** provides an argument containing the instance of the class **1034** upon which the view-class **1516** operates. A step **1520** writes the categorical class forward declaration **1492** using the id **1022** of the class **1518**. A step **1522** accumulates the categorical class forward declaration **1492** to the forward class declarations **1488**. A step **1524** dispatches the members **1038** of the class **1518**. A step **1526** returns boolean true, indicating success in processing the supplied class **1518**.

[0218] Refer to FIG. **73**. A view-handler view-member-datum **1528** receives an instance of the member-datum **1050**, and operates on the include directives **1494**, the typedef definitions **1496**, the typedef definition **1308**, and the include requirement **1310**. A datum member-datum **1530** provides an argument containing the instance of the member-datum **1050** upon which the view-member-datum **1528** operates. A step **1532** delegates to the type **1046** of the member-datum **1530**. A step **153.4** accumulates the include requirement **1310** of the delegate to the include directives **1494**. A step **1536** accumulates the typedef definition **1308** of the delegate to the typedef definitions **1496**. A step **1538** returns boolean true, indicating success in processing the supplied member-datum **1530**.

[0219] Refer to FIG. **74**. A view-handler view-host-class **1540** receives an instance of the host-class **1040**, and operates on the forward class declarations **1488** and the host forward declaration **1498**. A datum host-class **1542** provides

an argument containing the instance of the host-class **1040** upon which the view-host-class **1540** operates. A step **1544** writes the host forward declaration **1498** using the id **1022** of the host-class **1542**. A step **1546** accumulates the host forward declaration **1498** to the forward class declarations **1488**. A step **1548** returns boolean true, indicating success in processing the supplied host-class **1542**.

4.3 Member Datum Writer View Operant

[0220] Refer to FIG. **75**. A view-operant member datum writer **1550** processes an instance of the member-datum **1050**. The member datum writer **1550** has base the meta-module-viewer **1018**. A datum member datum definition **1552** provides the definition for the specified datum. A datum datum initializer **1554** defines an initial value for the specified datum. A datum datum cleanup **1556** defines a cleanup operation for the specified datum. A datum member meron view dispatch operation **1558** defines a dispatch to view a meron corresponding to the specified datum. A datum member meron edit dispatch operation **1560** defines a dispatch to edit a particular meron of a specified class. A datum view access definition **1562** defines a view access member function for a specified datum. A datum view access definitions **1564** collects member functions for view access to member data. A datum reset access definition **1566** defines a reset access member function for a specified datum. A datum reset access definitions **1568** collects member functions for reset access to member data. A datum edit access definition **1570** defines an edit access member function for a specified datum. A datum edit access definitions **1572** collects member functions for edit access to member data.

[0221] Refer to FIG. **76**. A view-entry member-datum-entry **1574** receives an instance of the member-datum **1050**, and operates on the member datum definition **1552** and the type identifier **1304**. A datum member-datum **1576** provides an argument containing the instance of the member-datum **1050** upon which the member-datum-entry **1574** operates. A step **1578** delegates to the type **1046** of the member-datum **1576**. A step **1580** writes the member datum definition **1552** using the scope-id **1044** of the member-datum **1576** and the type identifier **1304** of the delegate. A step **1582** dispatches the type **1046** of the member-datum **1576**.

[0222] Refer to FIG. **77**. A view-handler view-value-type **1584** receives an instance of the value-type **1056**, and operates on the view access definitions **1564**, the view access definition **1562**, the reset access definitions **1568**, the reset access definition **1566**, and the type identifier **1304**. A datum value-type **1586** provides an argument containing the instance of the value-type **1056** upon which the view-value-type **1584** operates. A step **1588** writes the view access definition **1562** using the scope-id **1044** of the datum and the type identifier **1304** of the delegate. A step **1590** accumulates the view access definition **1562** to the view access definitions **1564**. A step **1592** writes the reset access definition **1566** using the scope-id **1044** of the datum and the type identifier **1304** of the delegate. A step **1594** accumulates the reset access definition **1566** to the reset access definitions **1568**. A step **1596** returns boolean true, indicating success in processing the supplied value-type **1586**.

[0223] Refer to FIG. **78**. A view-handler view-reference-type **1598** receives an instance of the reference-type **1062**, and operates on the datum initializer **1554**, the datum cleanup **1556**, the view access definitions **1564**, the view access definition **1562**, the edit access definitions **1572**, the edit access definition **1570**, the reset access definitions **1568**, the reset access definition **1566**, the type identifier **1304**, the member meron view dispatch operation **1558**, and the member meron edit dispatch operation **1560**. A datum reference-type **1600** provides an argument containing the instance of the reference-type **1062** upon which the view-reference-type **1598** operates. A step **1602** writes the view access definition **1562** using the scope-id **1044** of the datum and the type identifier **1304** of the delegate. A step **1604** accumulates the view access definition **1562** to the view access definitions **1564**. A step **1606** writes the edit access definition **1570** using the scope-id **1044** of the datum and the type identifier **1304** of the delegate. A step **1608** accumulates the edit access definition **1570** to the edit access definitions **1572**. A step **1610** writes the reset access definition **1566** using the scope-id **1044** of the datum and the type identifier **1304** of the delegate. A step **1612** writes the datum initializer **1554**. A step **1614** tests for is-meron **1052** of the datum. A step **1616** writes the datum cleanup **1556** using the scope-id **1044** of the datum. A step **1618** writes the reset access definition **1566** using the scope-id **1044** of the datum. A step **1620** writes the member meron view dispatch operation **1558** using the scope-id **1044** of the datum. A step **1622** writes the member meron edit dispatch operation **1560** using the scope-id **1044** of the datum. A step **1624** accumulates the reset access definition **1566** to the reset access definitions **1568**. A step **1626** returns boolean true, indicating, success in processing the supplied reference-type **1600**.

[0224] Refer to FIG. **79**. A view-handler view-compound-type **1628** receives an instance of the compound-type **1066**, and operates on the datum cleanup **1556**, the view access definitions **1564**, the view access definition **1562**, the edit access definitions **1572**, the edit access definition **1570**, and the type identifier **1304**. A datum compound-type **1630** provides an argument containing the instance of the compound-type **1066** upon which the view-compound-type **1628** operates. A step **1632** writes the view access definition **1562** using the scope-id **1044** of the datum and the type identifier **1304** of the delegate. A step **1634** accumulates the view access definition **1562** to the view access definitions **1564**. A step **1636** writes the edit access definition **1570** using the scope-id **1044** of the datum and the type identifier **1304** of the delegate. A step **1638** accumulates the edit access definition **1570** to the edit access definitions **1572**. A step **1640** tests for is-meron **1052** of the datum. A step **1642** writes the datum cleanup **1556** using the scope-id **1044** of the datum. A step **1644** returns boolean true, indicating success in processing the supplied compound-type **1630**.

[0225] Refer to FIG. **80**. A view-handler view-reference-sequence-type **1646** receives an instance of the reference-sequence-type **1074**, and operates on the member meron view dispatch operation **1558** and the member meron edit dispatch operation **1560**. A datum reference-sequence-type **1648** provides an argument containing the instance of the reference-sequence-type **1074** upon which the view-reference-sequence-type **1646** operates. A step **1650** tests for is-meron **1052** of the datum. A step **1652** writes the member meron view dispatch operation **1558** using the scope-id **1044** of the datum. A step **1654** writes the member meron edit dispatch operation **1560** using the scope-id **1044** of the datum. A step **1656** returns boolean true, indicating success in processing the supplied reference-sequence-type **1648**.

[0226] Refer to FIG. **81**. A view-handler view-reference-set-type **1658** receives an instance of the reference-set-type

1080, and operates on the member meron view dispatch operation **1558** and the member meron edit dispatch operation **1560**. A datum reference-set-type **1660** provides an argument containing the instance of the reference-set-type **1080** upon which the view-reference-set-type **1658** operates. A step **1662** tests for is-meron **1052** of the datum. A step **1664** writes the member meron view dispatch operation **1558** using the scope-id **1044** of the datum. A step **1666** writes the member meron edit dispatch operation **1560** using the scope-id **1044** of the datum. A step **1668** returns boolean true, indicating success in processing the supplied reference-set-type **1660**.

4.4 Class Writer View Operant

[0227] Refer to FIG. **82**. A view-operant class writer **1670** processes an instance of the class **1034**.

[0228] The class writer **1670** has base the meta-module-viewer **1018**. A datum categorical class definition **1672** defines a categorical class for the specified module. A datum class view definition **1674** defines a member function of the viewer, responsive to an instance of a categorical class, dispatching views corresponding to its merons and base classes, if any. A datum class edit definition **1676** defines a member function of the editor, responsive to an instance of a categorical class, dispatching edits corresponding to its merons and base classes, if any. A datum meron view dispatch operations **1678** defines a collection of dispatch operations to the merons of a specified class, if any. A datum meron edit dispatch operations **1680** defines a collection of dispatch operations to the merons of a specified class, if any. A datum member data definitions **1682** defines the members of the class. A datum base class view dispatch operation **1684** defines a dispatch corresponding to a particular base class of a specified class. A datum base class view dispatch operations **1686** defines a collection of dispatches to views corresponding to the base classes of a specified class, if any. A datum view dispatch **1688** defines a member function of a categorical class to dispatch a view by a viewer. A datum base class edit dispatch operation **1690** defines a dispatch corresponding to a particular base class of a categorical class. A datum base class edit dispatch operations **1692** defines a collection of dispatches to edits corresponding to the base classes of a categorical class, if any. A datum edit dispatch **1694** defines a member function of the categorical class to dispatch an edit by an editor. A datum initializations **1696** defines expressions for initialization of member data. A datum cleanup expressions **1698** defines expressions for cleanup of member data.

[0229] Refer to FIG. **83**. A view-entry class-entry **1700** receives an instance of the class **1034**, and operates on the categorical class definition **1672**, the member data definitions **1682**, the view dispatch **1688**, the edit dispatch **1694**, the class view definition **1674**, the meron view dispatch operations **1678**, the base class view dispatch operations **1686**, the base class view dispatch operation **1684**, the class edit definition **1676**, the meron edit dispatch operations **1680**, the base class edit dispatch operations **1692**, and the base class edit dispatch operation **1690**. A datum class **1702** provides an argument containing the instance of the class **1034** upon which the class-entry **1700** operates. A step **1704** dispatches the members **1038** of the class **1702**. A step **1706** accumulates the meron view dispatch operations **1678** to the class view definition **1674**. A step **1708** accumulates the meron edit dispatch operations **1680** to the class edit definition **1676**. A step **1710** accumulates the member data definitions **1682** to the categorical class definition **1672**. A step **1712** iterates a genus class over each of the genera **1036** of the class **1702**. A step **1714** writes the base class view dispatch operation **1684** using the genus class. A step **1716** accumulates the base class view dispatch operation **1684** to the base class view dispatch operations **1686**. A step **1718** accumulates the base class view dispatch operations **1686** to the class view definition **1674**. A step **1720** writes the view dispatch **1688**. A step **1722** accumulates the view dispatch **1688** to the categorical class definition **1672**. A step **1724** iterates a genus class over each of the genera **1036** of the class **1702**. A step **1726** writes the base class edit dispatch operation **1690** using the genus class. A step **1728** accumulates the base class edit dispatch operation **1690** to the base class edit dispatch operations **1692**. A step **1730** accumulates the base class edit dispatch operations **1692** to the class edit definition **1676**. A step **1732** writes the edit dispatch **1694**. A step **1734** accumulates the edit dispatch **1694** to the categorical class definition **1672**.

[0230] Refer to FIG. **84**. A view-handler view-member-datum **1736** receives an instance of the member-datum **1050**, and operates on the member data definitions **1682**, the member datum definition **1552**, the datum initializer **1554**, the datum cleanup **1556**, the initializations **1696**, the cleanup expressions **1698**, the meron view dispatch operations **1678**, the member meron view dispatch operation **1558**, and the member meron edit dispatch operation **1560**. A datum member-datum **1738** provides an argument containing the instance of the member-datum **1050** upon which the view-member-datum **1736** operates. A step **1740** delegates to the member-datum **1738**. A step **1742** accumulates the member datum definition **1552** of the delegate to the member data definitions **1682**. A step **1744** accumulates the datum initializer **1554** of the delegate to the initializations **1696**. A step **1746** accumulates the datum cleanup **1556** of the delegate to the cleanup expressions **1698**. A step **1748** accumulates the member meron view dispatch operation **1558** of the delegate to the meron view dispatch operations **1678**. A step **1750** returns boolean true, indicating success in processing the supplied member-datum **1738**.

4.5 Module Writer View Operant

[0231] Refer to FIG. **85**. A view-operant module writer **1752** processes an instance of the module **1024**. The module writer **1752** has base the meta-module-viewer **1018**. A datum module definition **1754** provides class definitions for the specified module. A datum viewer class definition **1756** defines a viewer class for the specified module. The viewer class definition **1756** provides a base class suitable for specialization to process a domain-specific object-oriented data structure corresponding to the specified module. A datum class view definitions **1758** collects the class view definition **1674** member functions. A datum editor class definition **1760** defines an editor class for the specified module. The editor class definition **1760** provides a base class suitable for specialization to process a domain-specific object-oriented data structure corresponding to the specified module. A datum class edit definitions **1762** collects the class edit definition **1676** member functions. A datum categorical class definitions **1764** defines the categorical classes for the specified module. A datum host class definition **1766** defines the host class for the specified module.

[0232] Refer to FIG. **86**. A view-entry module-entry **1768** receives an instance of the module **1024**, and operates on the viewer class definition **1756**, the class view definitions **1758**, the editor class definition **1760**, and the class edit definitions **1762**. A datum module **1770** provides an argument containing the instance of the module **1024** upon which the module-entry **1768** operates. A step **1772** dispatches the host **1028** of the module **1770**. A step **1774** dispatches the classes **1026** of the module **1770**. A step **1776** accumulates the class view definitions **1758** to the viewer class definition **1756**. A step **1778** accumulates the class edit definitions **1762** to the editor class definition **1760**.

[0233] Refer to FIG. **87**. A view-handler view-class **1780** receives an instance of the class **1034**, and operates on the categorical class definitions **1764**, the categorical class definition **1672**, the class view definitions **1758**, the class view definition **1674**, the class edit definitions **1762**, and the class edit definition **1676**. A datum class **1782** provides an argument containing the instance of the class **1034** upon which the view-class **1780** operates. A step **1784** delegates to the class **1782**. A step **1786** accumulates the categorical class definition **1672** of the delegate to the categorical class definitions **1764**. A step **1788** accumulates the class view definition **1674** of the delegate to the class view definitions **1758**. A step **1790** accumulates the class edit definition **1676** of the delegate to the class edit definitions **1762**. A step **1792** returns boolean true, indicating success in processing the supplied class **1782**.

[0234] Refer to FIG. **88**. A view-handler view-host-class **1794** receives an instance of the host-class **1040**, and operates on the host class definition **1766** and the member data definitions **1682**. A datum host-class **1796** provides an argument containing the instance of the host-class **1040** upon which the view-host-class **1794** operates. A step **1798** delegates to the host-class **1796**. A step **1800** accumulates the member data definitions **1682** of the delegate to the host class definition **1766**. A step **1802** returns boolean true, indicating success in processing the supplied host-class **1796**.

5 Example Module Specification and Definition

[0235] A simple application demonstrates the production of a module definition from a module specification. The demonstration application recursively scans one or more file-system directories looking for media files. The media files which are detected are presented in a collection of interlinked HTML pages. The pages of the presentation reflect the directory organization of the scanned directories. Each discovered media file is presented in a hypertext link. The pages of the presentation are suitable for service by a web server running on a dedicated media device, such as a digital video recorder. Activation of a link presenting a particular media file generates a request to play the associated media on the dedicated media device. The application may form a potentially useful element for remote, web-oriented management of a dedicated media device.

[0236] The demonstration application uses a representation of media files in a file system. The representation uses an object-oriented module including a family of interrelated classes. The classes represent files and directories. The object-oriented module is specified using a module specification language.

5.1 Example Module Specification

[0237] Refer to FIG. **89**, which depicts a module specification using a module specification language, which is fully disclosed in a co-pending application. A module media-scanner **1804** represents media files arranged in a file system. The media-scanner **1804** corresponds to an instance of the module **1024**. A class host **1806** represents an instantiation of the module. The host **1806** corresponds to an instance of the host **1028**. A class viewer **1808** specifies a viewer class definition **1756** to be defined. This example does not specify a editor class definition **1760**.

[0238] The media-scanner **1804** specifies the following classes; these are examples of the classes **1026**. A class file **1810** represents a file in a file system, including regular files and directories. The file **1810** has no genera. A class directory **1812** represents a file system directory, potentially containing files, some of which may themselves be directories. The directory **1812** specializes the file **1810**. The specialization exemplifies the genera **1036**. A class top-directory **1814** represents a topmost directory from which a scan has been initiated. The top-directory **1814** specializes the directory **1812**. A class regular-file **1816** represents a regular file, i.e. a file which is not a directory. The regular-file **1816** specializes the file **1810**. A class audio-file **1818** represents an audio file. The audio-file **1818** specializes the regular-file **1816**. A class video-file **1820** represents a video file. The video-file **1820** specializes the regular-file **1816**.

[0239] A member datum root-files **1822**, specified for the host **1806**, specifies a set of instances of the file **1810**. The root-files **1822** represents the top-level collection of files in a scan. The root-files **1822** exemplifies the members **1038**. The root-files **1822** is specified as a meron, corresponding to the is-meron **1052**. Other member data include, for the file **1810**, a member datum parent **1824** and a member datum name **1826**. The parent **1824** represents the directory containing a particular file. The parent **1824** exemplifies the use of the reference-type **1062**. The name **1826** represents the name associated with a particular file in a particular directory. The name **1826** exemplifies the use of the text-type **1060**. For the directory **1812**, a member datum files **1828** is specified. The files **1828** representes the collection of files contained in a particular directory. The files **1828** exemplifies the use of the reference-set-type **1080**. The files **1828** also exemplifies meron specification corresponding to the is-meron **1052**.

5.2 Example Module Definition

[0240] Refer to FIG. **90**, which depicts an exemplary module definition for an object-oriented module suitable for representation of media files in a file system. The module definition **1754** is exemplified by a collection of definitions corresponding to the specification of the media-scanner **1804**. The include directives **1494** are exemplified by an include for the set. Forward definitions are provided for the module classes, include the categorical classes file **1810**, directory **1812**, etc., as well as the unitary classes host **1806** and viewer **1808**. The typedef definitions **1496**, are exemplified by a type definition for sets of files.

[0241] A definition corresponding to the host **1806** exemplifies the host class definition **1766**. The definition includes an example of the member datum definition **1552**, corresponding to the root-files **1822**. Also corresponding to the root-files **1822** are examples of the edit access definitions **1572** and the view access definitions **1564**. In the destructor for the host **1806**, an example of the cleanup expressions **1698** and of the datum cleanup **1556** may be seen, corresponding to the specification of the is-meron **1052** for the root-files **1822**.

[0242] Refer to FIG. **91**, which continues the depiction of an exemplary module definition for an object-oriented module suitable for representation of media files in a file system. Class definitions corresponding to the file **1810** and the directory **1812** exemplify the categorical class definitions **1764**.

[0243] The definition corresponding to the file **1810**, exemplifying the categorical class definition **1672**, includes member data definitions exemplifying the member data definitions **1682** and the member datum definition **1552**. The parent **1824** is exemplary of the reference-type **1830**. The name **1826** is exemplary of the value-type **1832** and the text-type **1834**. Examples of the view access definitions **1564** and the reset access definitions **1568** are defined for both data; the edit access definitions **1572** is defined only for the parent **1824** with the reference-type **1830**. An example of the view dispatch **1688** is exhibited. Examples of the datum initializer **1554** and the initializations **1696** are defined for the parent **1824**. An example of the view dispatch **1688** is also defined.

[0244] The definition corresponding to the directory **1812**, further exemplifies the categorical class definition **1672**. The genera **1036** of the directory **1812** is reflected in the public derivation from the file **1810**. The member data definition for the files **1828** exemplifies the reference-set-type **1836**. Access to the files **1828** exemplifies the edit access definitions **1572** and the view access definitions **1564**. An additional example of the view dispatch **1688** is exhibited. The consequence of the is-meron **1052** applicable to the files **1828** is exemplified in the destructor definition, where examples of the cleanup expressions **1698** and examples of the datum cleanup **1556** are exhibited.

[0245] The module definition **1754** includes additional categorical class definitions **1764**, but these are omitted since they add little to the present examples.

[0246] Refer to FIG. **92**, which continues the depiction of an exemplary module definition for an object-oriented module suitable for representation of media files in a file system. A definition corresponding to the viewer **1808** is shown, exemplary of the viewer class definition **1756**. For each categorical class specified in the media-scanner **1804**, a class view definition **1674** is defined; together these make up the class view definitions **1758**. Examples are exhibited for the file **1810** and the directory **1812**; others are omitted since they add little to the present examples.

[0247] In each of the exemplary class view definitions **1758**, dispatches are made to subsidiary member functions corresponding to the merons and the base classes of the supplied argument. For the exemplary class view definition **1674** corresponding to the file **1810**, there are corresponding dispatches to member functions for the merons and bases, but in both cases these are empty hence relatively uninteresting. The exemplary class view definition **1674** corresponding to the directory **1812** leads to more interesting subsidiary operation. In the meron view, an exemplary meron view dispatch operations **1678**, and member meron view dispatch operation **1558** are exhibited. The meron view dispatches to the aggregate viewer operating on the files **1828**, which was designated by the is-meron **1052**. In the base view, an exemplary base class view dispatch operations **1686** is exhibited, dispatching to the class view definitions **1758** corresponding to the file **1810**, as specified by the genera **1036** for the directory **1812**.

6 Alternative Embodiments

[0248] The specializations of the value-type **1056** in the exemplary embodiment are representative but far from comprehensive. In an alternative embodiment, value types representing integers, floating point numbers, cardinal numbers, dates, times, geographical positions, etc., could be provided.

[0249] The specializations of the compound-type **1066** in the exemplary embodiment are sufficient for a wide range of applications, but additional compound types could be provided in an alternative embodiment. Compound types representing lists, heaps, multimaps, multisets, graphs, vectors, matrices, etc, could be provided in an alternative embodiment. Provision for compound types of compound types could also be provided in an alternative embodiment. The effect of nested compound types is readily obtained by providing categorical classes which simply contain the nested type as a member.

[0250] In the exemplary embodiments of the categorical classes and the operants, member data has been exhibited in a public scope. The exemplary embodiments of the operants utilize categorical class data accordingly. Exhibition of member data in a public scope enhances clarity and saves space in exposition. In an alternative embodiment, member data could be defined in a private scope, and access methods (member functions) could be provided to manipulate member data indirectly, in accordance with normal practice in object-oriented programming. In the demonstration module, ordinary practice is followed, and member data is exhibited in a private scope with edit, reset, and view access provided by member functions.

[0251] In the exemplary embodiments of the operants which process a module specification, error handling and reporting has been suppressed to enhance clarity and save space. In an alternative embodiment, boolean return values from dispatches and failure status from delegations could be examined and appropriate error handling and reporting could be provided. In another alternative embodiment, which is not preferred, error handling could be completely ignored, and void returns could be used for handlers.

[0252] In the exemplary embodiment, the module definition is presented in the C++ programming language. In an alternative embodiment, Java or C# could serve for module definition, with suitable modifications to accomodate differences in template programming, inheritance, etc. Other target languages could be utilized with variable efforts, depending on the degree of support for inheritance and polymorphism in dispatch.

I claim:

1. A computer-implemented method of processing a module specification to produce a module definition, comprising the steps of:

viewing a module, said module included in said module specification,

dispatching a plurality of categorical classes, said categorical classes included in said module,

accumulating a plurality of class view definitions to a viewer class definition, said viewer class definition included in said module definition, and

viewing a categorical class, said categorical class included in said categorical classes and said categorical class optionally associated with one or more genus classes,

dispatching a plurality of class members, said class members included in said categorical class,

viewing a datum, said datum included in said class members,

writing a datum definition,

accumulating said datum definition to a plurality of member data definitions,

testing for a meron qualification, said meron qualification included in said datum,

according to the success of said test for said meron qualification, writing a member meron view dispatch operation, and

accumulating said member view meron dispatch operation, if any, to a plurality of meron view dispatch operations,

accumulating said meron view dispatch operations to a class view definition,

accumulating said member data definitions to a categorical class definition,

iterating a genus class over each of said genus classes, if any,

writing a base class view dispatch operation using said genus class, and

accumulating said base class view dispatch operation to a plurality of base class view dispatch operations,

accumulating said base class view dispatch operations, if any, to said class view definition,

writing a view dispatch,

accumulating said view dispatch to said categorical class definition,

accumulating said categorical class definition to a plurality of categorical class definitions, said categorical class definitions included in said module definition, and

accumulating said class view definition to said class view definitions.

* * * * *