Projektziel

Ziel dieses Projekts ist es, ein Application Programming Interface (API) in C++ zu entwicklen, mit dem einfach C Source-Code generiert werden kann. Die Generierung einer C-Funktion soll in einem eigenen API gekapselt werden. Der generierte Source-Code soll zum einen mit Kommentaren im Doxygen-Format und ohne Kommentare möglich sein.

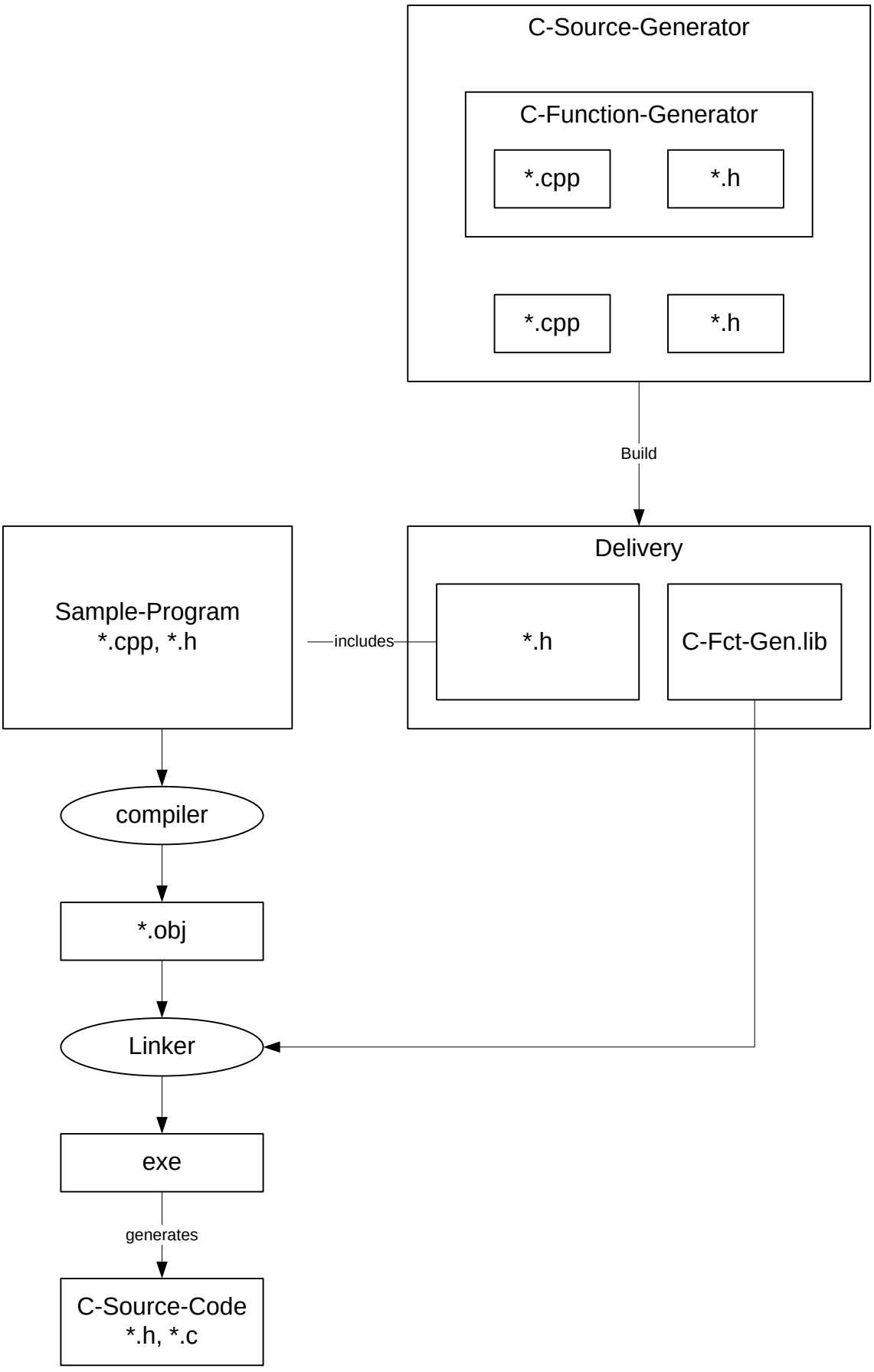
Grundlage sind die Ergebnisse des gleichen Projekts aus dem Wintersemester 2010/2011. Hier müssen die einzelnen Klassen überarbeitet werden. Unter Umständen ist auch eine Neuentwicklung notwendig.

Optional: Eine auf Qt-basierte Oberfläche für einen Teil der Funktionalität.

Projektarchitektur

Die Funktionalität der Souce-Code Generierung soll in einem eigenen Paket geschehen. Diese Funktionalität wird in Form einer statischen Bibliothek (cscg.lib bzw. cscg.ar) und einer Liste von Headerdateien zur Verfügung gestellt.

Workflow


C-Source-Generator
C-Function-Generator
*.cpp
*.h
*.cpp
*.h
Build
Delivery
Sample-Program
*.cpp, *.h
includes
*.h
C-Fct-Gen.lib
compiler
*.obj
Linker
exe
generates
C-Source-Code
*.h, *.c

API Design-Vorschläge

Modul

```
Modul m("MyModul");
RawOut ro;
ro << m;
DoxygenOut do;
do << m;
m.generate();
```

```
/* filename: MyModul.h */
#ifndef _MY_MODUL_H_
#define _MY_MODUL_H_
/* Deklarationen */
#endif /* _MY_MODUL_H_ */
```

```
/* filename: MyModul.c */
#include "MyModul.h"
/* Definitionen */

#ifdef UNIT_TEST_MY_MODUL
int main(int argc, char** argv)
{
      int errorCount = 0;
      /* test cases */
      return errorCount;
}
#endif
```

Modul-Komponenten

Funktionen

```
Modul m("MyModul");
Function f1("help");
f1.setVisibility(PRIVATE);

Function f2("interface");
f2.setReturnType("int");

m.add(f1);
m.add(f2);

RawOut ro;
ro << m;
//m.generate();
```

```
/* filename: MyModul.h */
#ifndef _MY_MODUL_H_
#define _MY_MODUL_H_

/* Schnittselle -
   Funktionsdeklarationen */
int interface();

#endif /* _MY_MODUL_H_ */
```

```
/* filename: MyModul.c */
#include "MyModul.h"

/* Deklarationen der
Hilfsfunktionen*/
static void help();

/* Definition der Schnittstelle */
int interface()
{
}

/* Definition der Hilfsfunktionen */
static void help()
{
}
```

Variablen

```
Modul m("MyModul");
Variable v1("global", "int", "0");
Constant v2("KONST", "int", "2");
Variable v3("intern", "int", "1", PRIVATE);
Constant v4("INTERN", "int", "3", PRIVATE);

m.add(v1); m.add(v2); m.add(v3); m.add(v4);

RawOut ro;
ro << m;
//m.generate();
```

```
/* filename: MyModul.h */
#ifndef _MY_MODUL_H_
#define _MY_MODUL_H_

/* Deklaration
   globale Variablen */
extern int global;

/* globale Konstanten */
static const int KONST = 2;
/* #define KONST 2 */

#endif /* _MY_MODUL_H_ */
```

```
/* filename: MyModul.c */
#include "MyModul.h"

/* Globale Variablen */
int global = 0;

/* Interne globale Variablen -
Zustaende */
static int intern = 1;

/* Interne Konstanten */
static const int INTERN = 3;
```

Enumeration types

```
Enumeration e("Himmelsrichtung");
e.add("NORDEN", "1");
e.add("SUEDEN", "2");
e.add("WESTEN", "3");
e.add("OSTEN", "4");
string dcl = e.getDeclaration();
```

```
typedef enum
{
      NORDEN = 1,
      SUEDEN,
      WESTEN,
      OSTEN
} Himmelsrichtung;
```

Strukturen

```
Structur s("Zylinder");
s.addComponent("hoehe", "double");
s.addComponent("radius", "double");
string dcl = e.getDeclaration();
```

```
typedef struct Zylinder
{
      double hoehe;
      double radius;
} Zylinder;
```

Validierung

Unter Benutzung des Frameworks soll ein Programm geschrieben werden, das vom konkreten Beispiel „CylinderWorkspace“ ein Source-Code-Gerüst generiert. Ein Ausschnitt des Source-Codes könnte sein:

```
Int main()
{
      Programm p("CylinderApplication");
      Modul cylinderIO("cylinderIO");
      p.add(cylinderIO);

      Library l("CylinderLibray");
      Modul cylinder("cylinder");
      Modul circle("circle");
      l.add(cylinder);
      l.add(circle);

      Workspace ws("CylinderWorkspace");
      ws.addProject(p);
      ws.addProject(l);
      ws.generate(/* Doxygenformat */);

      return 0;
}
```