Interface Specification

TTK91 Simulator

version 1.0

TTK	Software Engineering	581206-4 Software Engineering Project
Author: Kalle Kärkkäinen		Printed: 15.3.2004
Distribution: Malan project group, Koski project group, Teemu Kerola, Antti Tevanlinna, Raine Kauppinen		
Document state: Final		Modified: 15.3.2004

VERSION CONTROL

Version	Date	Author	Description
1.0	27.02	Kärkkäinen	Created

TABLE OF CONTENTS

1. ABSTRACT

In spring 2004 two projects were started on the same task, creating a replacement for the old Koksi software that is used as a teaching tool in Computer Organization course. It was required that these two programs have a common interface.

This document describes that programming interface common to both project groups.

1.1 Purpose and scope

This document is intended to be used as a guide for each project group in implementing the common interface. The work for this document was done as collaboration between both project groups.

This document is delivered to Teemu Kerola for maintaining. This document is not part of either projects delivery.

1.2 Definitions terms and abbreviations

Term	Description
Malan	Other Project Group working on this issue
Koksi	A simulated computer environment
MS-DOS	Microsoft Disk Operating System
TTK91	Programming language specification
Koski	Other Project Group working on this issue
Java	Programming language developed by Sun Microsystems
OO	Object Oriented, current programming dogma

1.3 Overview

Chapter 1 Abstract describes this document. This chapter contains all the abbreviations used in the document.

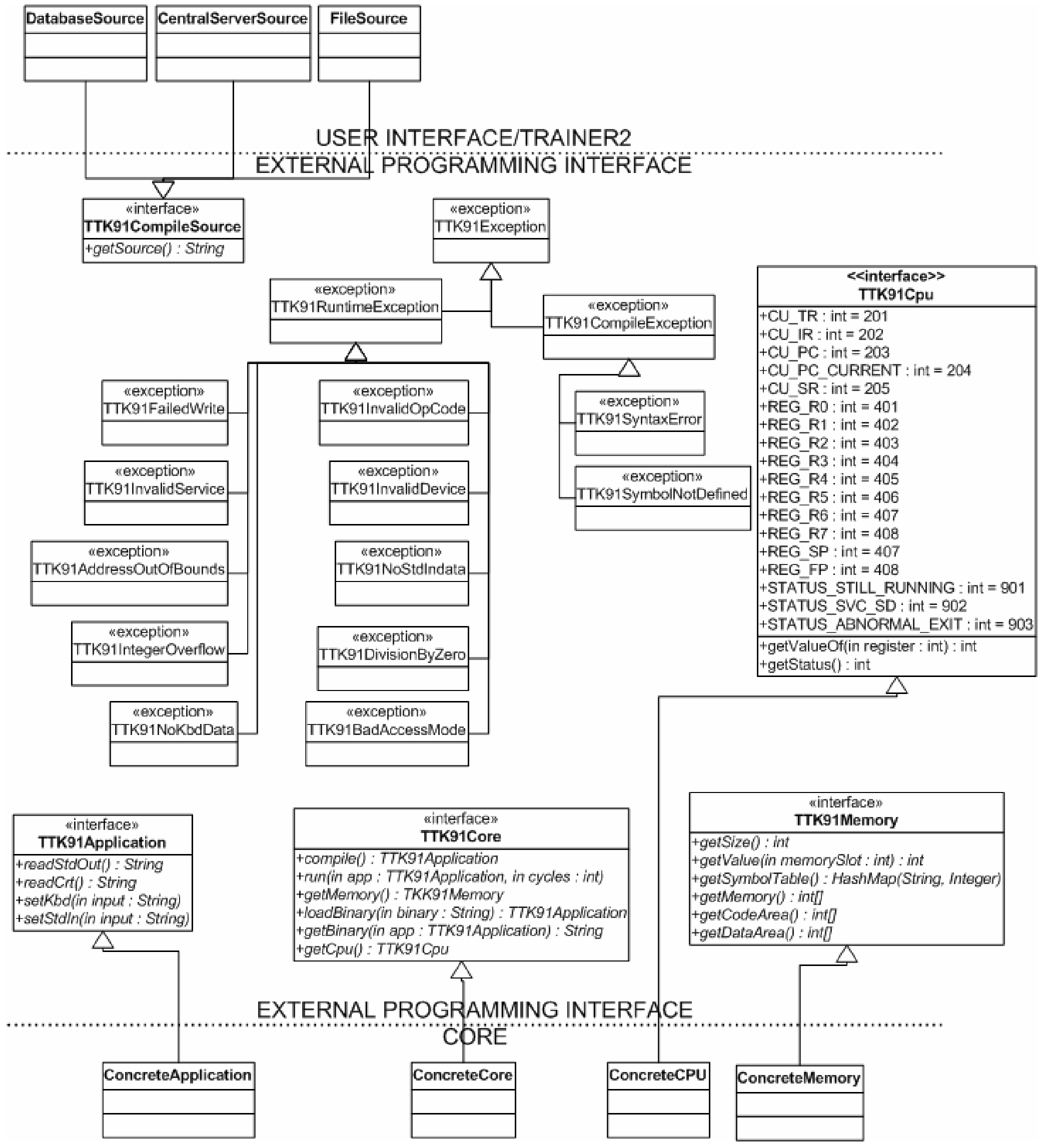
Chapter 2 Design explains the design in general detail.

Chapter 3 Classes shows the internal workings of the design.

Chapter 4 Changes describes the process used to make changes into this document.

2. DESIGN

2.1 UML class diagram


DatabaseSource
CentralServerSource
FileSource
USER INTERFACE/TRAINER2
EXTERNAL PROGRAMMING INTERFACE
«interface» TTK91CompileSource
+getSource() : String
«exception» TTK91Exception
«exception» TTK91RuntimeException
«exception» TTK91CompileException
«exception» TTK91FailedWrite
«exception» TTK91InvalidOpCode
«exception» TTK91SyntaxError
«exception» TTK91InvalidService
«exception» TTK91InvalidDevice
«exception» TTK91SymbolNotDefined
«exception» TTK91AddressOutOfBounds
«exception» TTK91NoStdIndata
«exception» TTK91IntegerOverflow
«exception» TTK91DivisionByZero
«exception» TTK91NoKbdData
«exception» TTK91BadAccessMode
<<interface>> TTK91Cpu
+CU_TR : int = 201
+CU_IR : int = 202
+CU_PC : int = 203
+CU_PC_CURRENT : int = 204
+CU_SR : int = 205
+REG_R0 : int = 401
+REG_R1 : int = 402
+REG_R2 : int = 403
+REG_R3 : int = 404
+REG_R4 : int = 405
+REG_R5 : int = 406
+REG_R6 : int = 407
+REG_R7 : int = 408
+REG_SP : int = 407
+REG_FP : int = 408
+STATUS_STILL_RUNNING : int = 901
+STATUS_SVC_SD : int = 902
+STATUS_ABNORMAL_EXIT : int = 903
+getValueOf(in register : int) : int
+getStatus() : int
«interface» TTK91Application
+readStdOut() : String
+readCrt() : String
+setKbd(in input : String)
+setStdIn(in input : String)
«interface» TTK91Core
+compile() : TTK91Application
+run(in app : TTK91Application, in cycles : int)
+getMemory() : TKK91Memory
+loadBinary(in binary : String) : TTK91Application
+getBinary(in app : TTK91Application) : String
+getCpu() : TTK91Cpu
«interface» TTK91Memory
+getSize() : int
+getValue(in memorySlot : int) : int
+getSymbolTable() : HashMap(String, Integer)
+getMemory() : int[]
+getCodeArea() : int[]
+getDataArea() : int[]
EXTERNAL PROGRAMMING INTERFACE
CORE
ConcreteApplication
ConcreteCore
ConcreteCPU
ConcreteMemory

Picture 1 UML diagram.

Picture 1 shows this interface is not large or complex. It is a simple set of classes that provide the needed set of features. These features contain the core functionalities. Each project group is free to extend and implement other functionalities as they see fit.

The diagram is divided into three segments. UI segment (top), interface segment (middle) and core segment (down). The middle segment is seen as the communication channel between the UI and core segments. This chapter describes the middle segment in general terms.

2.2 General description

This library is written in Java programming language and it is placed into fi.hu.cs.ttk91 package.

TTK91Core is the entry point to the system. This is the only functional class, others are just ways to show and describe data. TTK91Core does not force implementation details, but it defines proper input, output and behavior. It offers the following services:

- compiling a TTK91CompileSource
- running a TTK91Application
- creating a TTK91Application out of TTK91 binary format
- creating a binary representation of a TTK91Application Object
- access to the TTK91Cpu
- access to the TTK91Memory

TTK91CompileSource is meant as a carrier for the source code. There might be different ways to get the source code, and this is the class where such action is intended to take place. It offers only one service:

- fetching the source code

TTK91Application represents the application to the world. It has accessors to application's features.

- set input (keyboard and standard input)
- read output (printed to monitor or standard output)

TTK91Cpu describes the CPU of the virtual machine. This class has only one feature.

- able to fetch any of the specified registers

TTK91Memory describes the memory of the runtime state. This class has many features.

- querying the size of the memory
- returning the value of any memory slot
- returning the memory as a bulk integer array
- returning only a portion of the memory (memory range as seen after compilation, code block or data block)
- returning the symbol table as it is currently

TTK91Exception has two child classes that describe the kind of exceptions that can arise during the compilation sequence and in runtime. This does not cover the other exceptions that might arise. Those are described in more detail in chapter 3 Classes.

3. CLASSES

3.1 TTK91CompileSource

TTK91CompileSource has only one method.

```
public String getSource()
```

This function may not throw any exceptions, and should return a string containing valid TTK91 source code. The validity of the source code is not a requirement that should be checked. It is checked at the compilation stage.

3.2 TTK91Core

TTK91Core contains all the methods that enable the UI to use the core.

```
public TTK91Application compile (TTK91CompileSource
source) throws TTK91Exception,
TTK91CompileException
```

```
public void run(TTK91Application app, int steps)
throws TTK91Exception, TTK91RuntimeException
```

```
public TTK91Memory getMemory()
```

```
public TTK91Cpu getCpu()
```

```
public TTK91Application loadBinary(String binary)
```

```
public String getBinary(TTK91Application)
```

Method TTK91Application compile(TTK91CompileSource) throws TTK91Exception, TTK91CompileException (and all the defined child classes) and IllegalArgumentException.

IllegalArgumentException is thrown for invalid TTK91CompileSource Object.

TTK91CompileException is thrown when compilation of the TTK91CompileSource has failed for some reason.

TTK91Exception is stated to provide means for extension.

Method compile(TTK91CompileSource) is used to compile a TTK91CompileSource into a TTK91Application. If no exceptions are thrown, this method must return a fully functional TTK91Application that can be run in the run method of the same TTK91Core Object. Cross TTK91Core (compiled with the other project groups compile and run on the other's run method) compatibility is not guaranteed in any other way than via the binary format.

Method TTK91Application loadBinary(String) throws ParseException if the String parameter is not in the valid TTK91 binary format. If no exceptions are thrown, the method must return a TTK91Application Object ready to be run in the same Core object.

String getBinary(TTK91Application) returns the application in TTK91 binary format. This format is possible to be read back into a TTK91Application using loadBinary(String) of the TTK91Core.

Method void run(TTK91Application, int) can throw TTK91Exceptions, TTK91RuntimeExceptions (and all the defined child classes) and IllegalArgumentException.

InvalidArgumentException is thrown for invalid TTK91Application Object or an int less than zero.

TTK91Exceptions are thrown from many issues. These are not fixed at this point as the main reason for throwing TTK91Exceptions is the need for possible extensions of functionality. One fixed reason is too big application to fit into the memory of the virtual machine.

TTK91RuntimeExceptions arise from flaws in the running program or lack of input. Specific exceptions are described later in chapter 3.6.

If no Exceptions are thrown either the TTK91Application has run its course or all required CPU cycles has been passed successfully.

The methods TTK91Memory getMemory() and TTK91Cpu getCpu() both return the current state of the memory or the CPU as a copies of the corresponding objects. No changes made by running the program are reflected in the objects that have been requested earlier. These methods do not throw exceptions.

3.3 TTK91Application

TTK91Application represents the needed features of the application:

```
public String readStdOut()
public String readCrt()
public void setKbd(String input)
public void setStdIn(String fileContent)
```

This class is able to carry output and input for the application. It does so with methods String readStdOut(), String readCrt(), void setKbd(String) and void setStdIn(String). The set methods also parse the content of the desired input. If there is any error in the parameter, none of it is used. Parameter is a whole number; any other characters invalidate the stream. Parameter tokens are delimited using ' ', '\t', '\r', '\n', ',', '.', ':' or ';'. There may be infinite amount of delimiters between tokens. If there are any problems with the parameter, set methods will throw IllegalArgumentException.

Output of the application can be gathered after any CPU cycle. When output is asked from the TTK91Application it will empty the stream storage. These functions do not throw any exceptions.

3.4 TTK91Cpu

TTK91Cpu models the CPU of the virtual machine. This class gives access to copy of the CPU's registers.

```
public int getRegister(int)
```

```
public int getRuntimeStatus()
```

These functions access CPU registers. They do not throw any exceptions. Method int getRegister(int) returns the value stored in the certain register.

Method int getRuntimeStatus() returns the status of the application, meaning whether it has stopped gracefully, is still running or has ended in an error.

3.5 TTK91Memory

TTK91Memory models the general memory of the virtual machine. It enables querying the memory banks using a specific slot or asking for a data or code region. It also allows querying the symbol table. This way it is easy to ask for the value of a specific variable no matter what it's location in the memory is.

```
public int getSize()
```

```
public int getValue(int slot)
```

```
public int[] getMemory()
```

```
public int[] getCodeArea()
```

```
public int[] getDataArea()
```

```
public HashMap getSymbolTable()
```

Method int getSize() returns the size of the whole memory. This function will never throw exceptions.

Method int getValue(int) returns the number stored in a given memory slot. This function will throw IllegalArgumentException if given slot is < 0 or ≥ int getLength().

Method int[] getMemory() returns the whole memory as a copy. This function can not throw exceptions.

Method int[] getCodeArea() and int[] getDataArea() both return a fragment of the memory that was seen as data or code areas directly after the compilation. These will not return stacks or other memory than aforementioned. These areas are given as copies. These methods will not throw exceptions.

Method HasMap getSymbolTable() returns the symbol table related to the compiled application. It contains symbol names as keys in the hash linked with their locations in memory.

3.6 TTK91Exception

This system has quite a bit of exceptions. These exceptions cover the errors that might occur even if the interface is used correctly.

TTK91Exception defines the only method specific to these classes. As they derive from java.lang.Exception they have other qualities specified there in greater detail.

3.6.1 TTK91CompileException

TTK91CompileException arises from compilation issues.

3.6.1.1 TTK91SyntaxError

This is thrown when the given source code is not of correct TTK91 source code syntax.

3.6.1.2 TTK91SymbolNotDefined

This is thrown when a symbol used in the source code is left undefined during the compilation.

3.6.2 TTK91RuntimeException

TTK91RuntimeExceptions are thrown according to the status register of the virtual machines CPU.

3.6.2.1 TTK91AddressOutOfBounds

This exception is thrown when addressing a memory bank that is out of scope. This means memory address less than zero or equal or greater than the size of the memory.

3.6.2.2 TTK91InvalidOpCode

This exception is caused by an operation code that is not in the specification.

3.6.2.3 TTK91InvalidDevice

TTK91InvalidDevice is thrown if the program is trying to use nonexistent device or an existing device in a wrong way, e.g. by reading from the CRT device or writing to standard input.

3.6.2.4 TTK91InvalidService

This is thrown when calling nonexistent system services.

3.6.2.5 TTK91IntegerOverflow

Integer overflow is thrown when integer passes it's maximum or minimum limit.

3.6.2.6 TTK91BadAccessMode

This is thrown if the program is trying to use an invalid addressing mode.

3.6.2.7 TTK91DivisionByZero

This is thrown when program is trying to divide something by zero.

3.6.2.8 TTK91NoStdInData

This is thrown when program tries to read from standard input pipe, when there is no data to read.

3.6.2.9 TTK91NoKbdData

This is thrown when program tries to read from keyboard pipe, when there is no data to read.

3.6.2.10 TTK91FailedWrite

This exception is the result of the program trying to write something to the monitor or to the standard output and the attempt failed.

4. CHANGES

This document is intended to ease the need for group-to-group collaboration.

4.1 Making changes

Changes that are necessary may arise. If this happens, Teemu Kerola is to be notified. No changes into this design may happen without this.

Contacting Mr. Kerola happens by a document. This document is called Change Request, and it must explain the following facts.

1. Why should the change occur (under title Problem Description)?
2. What should be changed (Problem Description)?
3. How it should be changed (Fix Suggestion)?
4. Are other parts of the design affected (Impact to Design)?

Although this is a formal document, it does not need to contain abstract or other document control information.

This document is intended to ease the need for project group to project group collaboration and communication.

Mr. Kerola is required to show the Change Request to the other project group before making his decision. The other project group may then:

1. refuse this change
2. suggest another change that would fix the problem
3. accept the change with no comments

Mr. Kerola will then evaluate each of these demands and come to a conclusion that is passed to both project groups along with a new version of this document.