CRAVE for UVM-SystemC

Daniel Große

Johannes Kepler University Linz, Austria
DFKI Bremen, Germany

crave@systemc-verification.org

Presentation on July 8th 2020
at VWG @ Accellera Systems Initiative

Randomization in UVM-SystemVerilog

- Basic randomization via $urandom() / $urandom_range() / $srandom()
- Constraints
  - Expressions to be satisfied by a constraint solver while creating values
  - Hard & soft constraints
  - Specification (if/else, foreach, solve/before, dist, inside, inline constraints)
- randomize()
  - causes new values to be selected according to their constraints within an object/class and its components
- Random variables (rand / randc)
- Randsequence (Grammar defined randomized sequence)

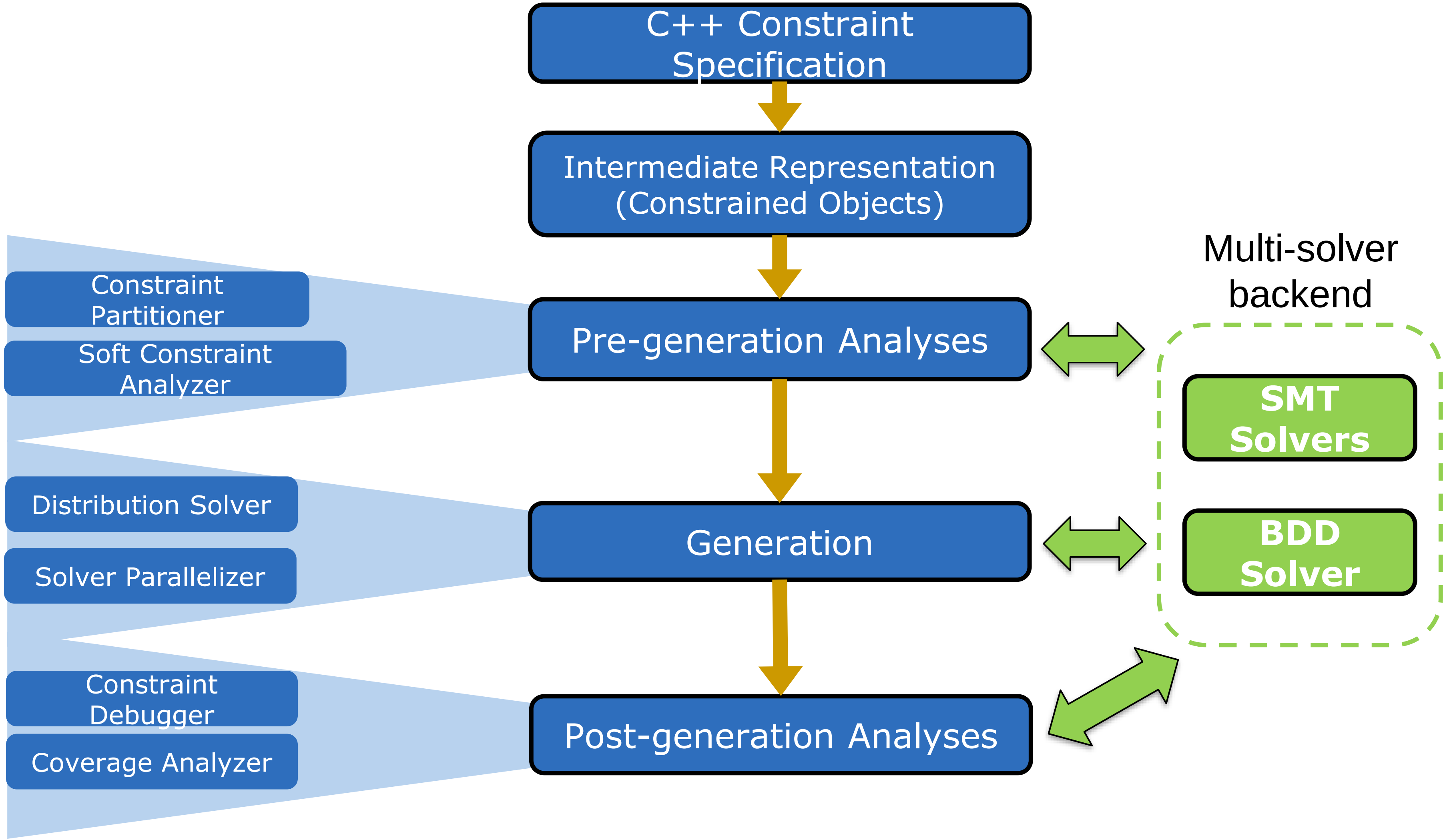
Randomization in UVM-SystemC

- SystemC Verification Library (SCV)
  - Hard and soft constraints
  - Weighted distribution
  - Biased distribution
  - Outdated non-scalable constraint solving technology based on *Binary Decision Diagrams* (BDDs)
- CRAVE
  - Powerful & extensible constrained random stimuli generator
  - Leverage latest constraint solving technologies
  - C++11 syntax for constraint definition

Basics of CRAVE

- **C**onstrained **RA**ndom **V**erification **E**nvironment
- Syntax and semantics closely followed SystemVerilog IEEE 1800 std
- Random objects
- Random variables
- Hard/soft constraints
- Efficient constraint solvers
- MIT license
- www.systemc-verification.org/crave

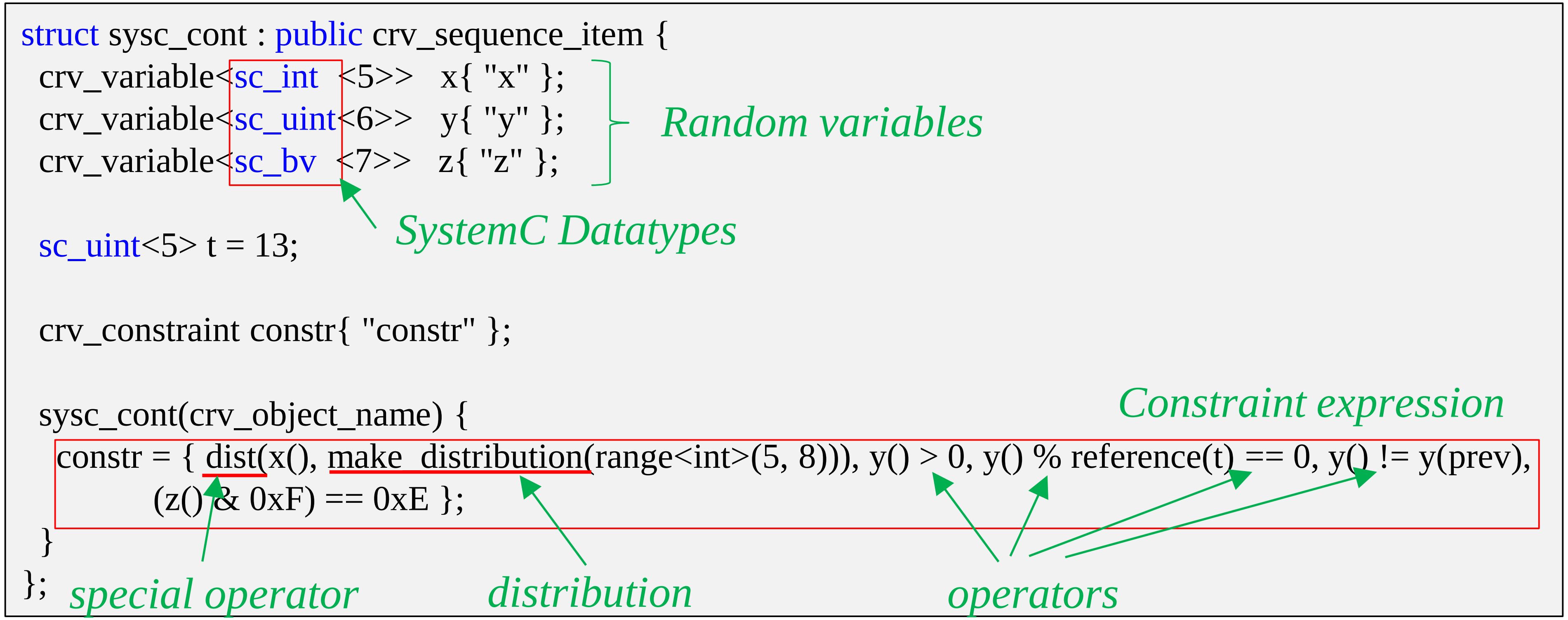
CRAVE Architecture



How-to-guide: Installation

1. If you already have UVM-SystemC installed and UVM_SYSTEMC_HOME is set, then jump to Step 4.
2. Check out the current OSCI-WG/uvm-systemc master branch
3. cd uvm-systemc-master/
4. tar xvfz crave2uvm_<date>_<time>.tar.gz   *[extracts crave into contrib/crave]*
5. cd contrib/crave
6. ./buildscript install   *[Please also read the README.md]*
7. ./buildscript compile   *[compile examples, i.e. ubus]*
8. ./buildscript run   *[run the ubus example with randomization]*

Randomization using CRAVE – Example



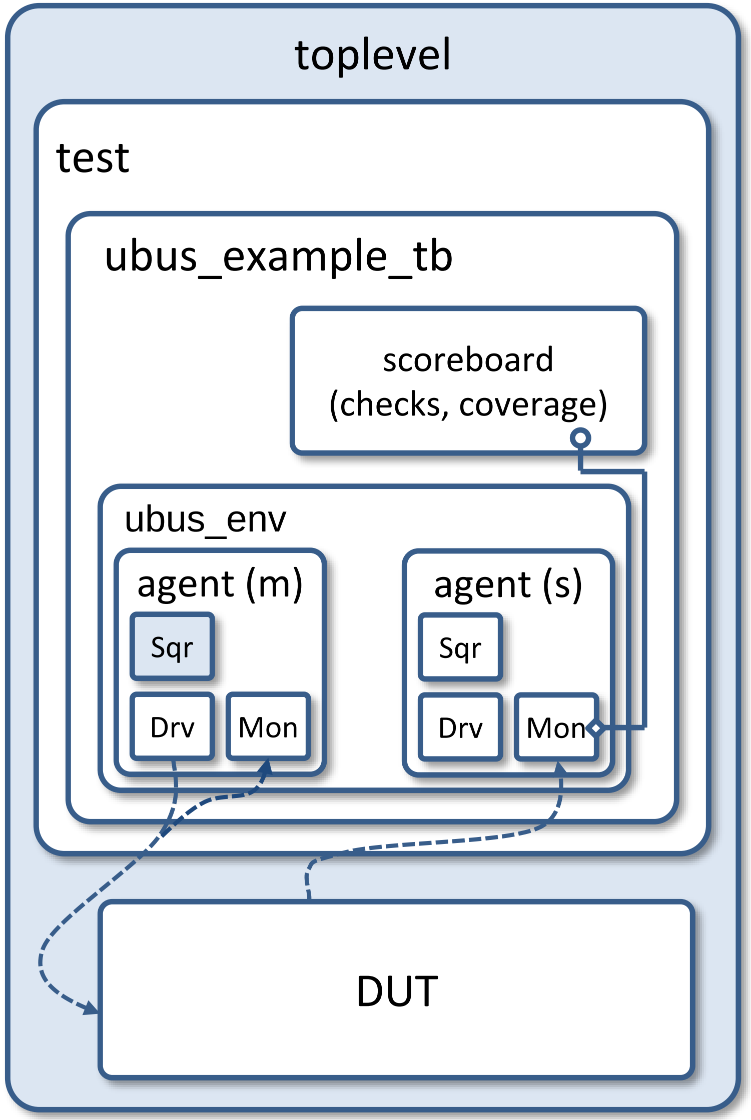
```
struct sysc_cont : public crv_sequence_item {
  crv_variable<sc_int  <5>>   x{ "x" };
  crv_variable<sc_uint<6>>   y{ "y" };
  crv_variable<sc_bv   <7>>   z{ "z" };

  sc_uint<5> t = 13;

  crv_constraint constr{ "constr" };

  sysc_cont(crv_object_name) {
    constr = { dist(x(), make_distribution(range<int>(5, 8))), y() > 0, y() % reference(t) == 0, y() != y(prev),
               (z() & 0xF) == 0xE };
  }
};
```

Special operators

- inside
- dist
- if_then
- if_then_else
- foreach
- unique
- bitslice

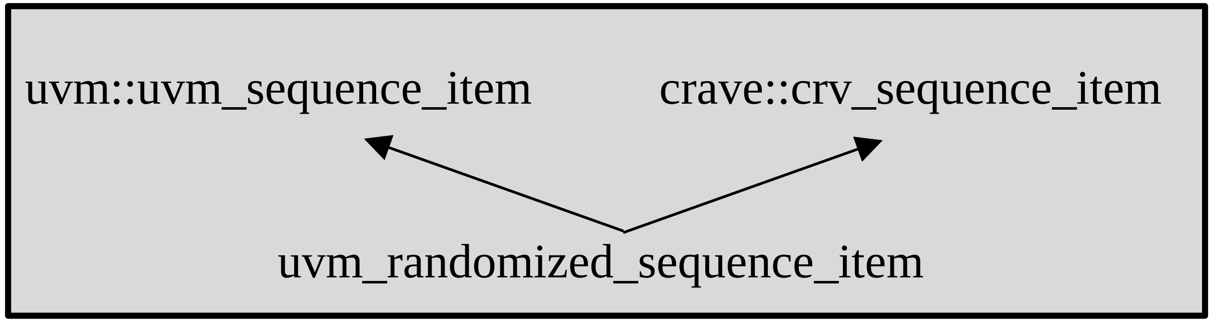
Intro to UBus

- Ubus
  - simple non-multiplexed
  - Synchronous
  - no pipelining
  - Address bus: 16 bit wide
  - Data bus: 8 bit wide
- N: number of Masters & Slaves supported
- Three sequential phase data transfer
  - Arbitration
  - Address
  - data
- UVM example provided in the UVM Users Guide (http://accellera.org/downloads/standards/uvm)



UBus – Sequence Item

Multiple inheritance



UBUS – sequence item

ubus_transfer.h

```
#include <crave2uvm.h>
....
class ubus_transfer : public uvm_randomized_sequence_item
{
public:
  crave::crv_variable< sc_dt::sc_uint<16> > addr;
  crave::crv_variable< ubus_read_write_enum> read_write;
  crave::crv_variable< unsigned int> size;
```

crave2uvm.h

```
#include "uvm_randomized_sequence_item.h"
#include "uvm_randomized_sequence.h‟

#undef UVM_DO
#define UVM_DO(SEQ_OR_ITEM) \
UVM_DO_ON_PRI_WITH(SEQ_OR_ITEM, this->m_sequencer, -1,)
```

uvm_randomized_sequence_item.h

```
class uvm_randomized_sequence_item:
                public crave::crv_sequence_item,
                public uvm::uvm_sequence_item
{
public:
  UVM_OBJECT_UTILS(uvm_randomized_sequence_item);
  uvm_randomized_sequence_item() :
                crave::crv_sequence_item(), uvm::uvm_sequence_item()
                { }
```

UBus – Sequence Item

File: examples/ubus/vip/
ubus_transfer.h

```
class ubus_transfer extends uvm_sequence_item;

  rand bit [15:0]           addr;
  rand ubus_rw_enum         read_write;
  rand int unsigned         size;
  rand bit [7:0]            data[];
  rand bit [3:0]            wait_state[];
  rand int unsigned         error_pos;
  rand int unsigned         transmit_delay = 0;
  ...

  constraint c_read_write {
    read_write inside { READ, WRITE };
  }
  constraint c_size {
    size inside {1,2,4,8};
  }
  constraint c_data_wait_size {
    data.size() == size;
    wait_state.size() == size;
  }
  constraint c_transmit_delay {
    transmit_delay <= 10 ;
  }
```

SystemVerilog

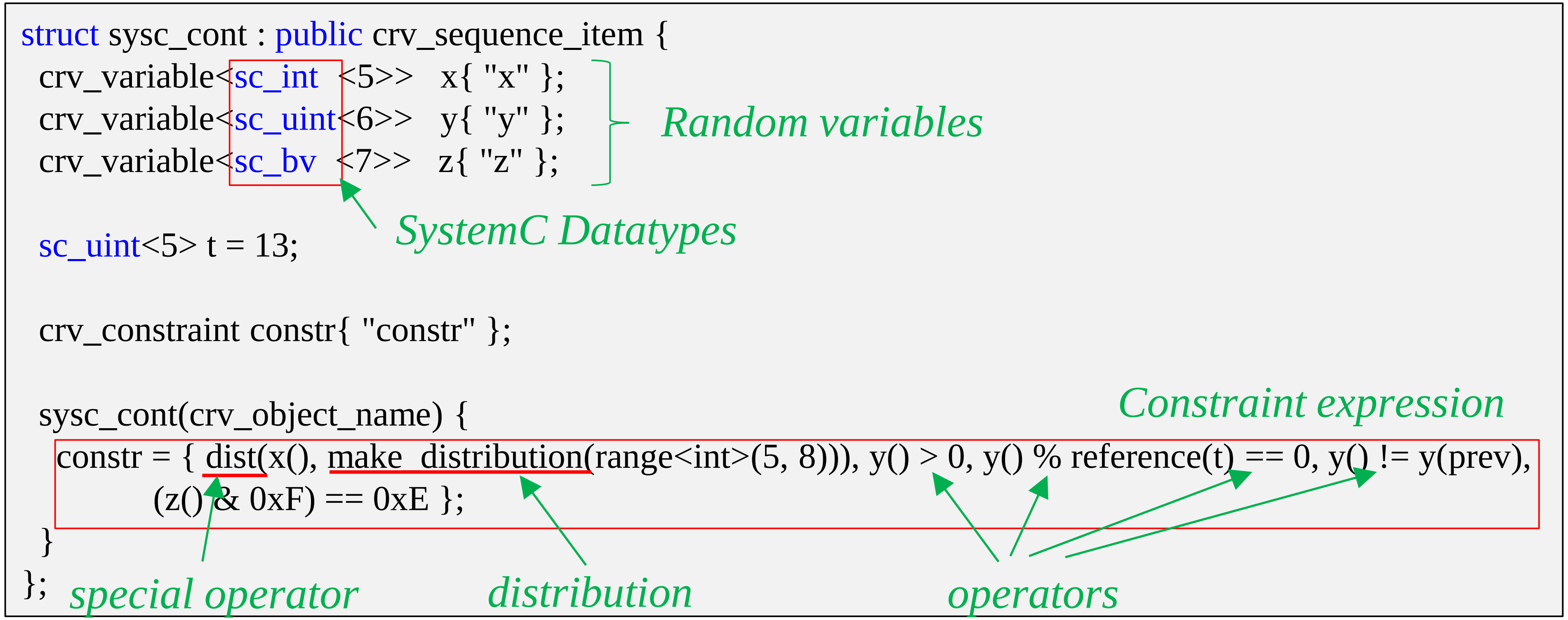
```
class ubus_transfer : public uvm_randomized_sequence_item {
public:
  crv_variable<ubus_rw_enum> read_write;
  crv_variable<sc_uint<16>> addr;
  crv_variable<unsigned> size;
  crv_vector<sc_bv<8>> data;
  crv_vector<sc_bv<4>> wait_state;
  crv_variable<unsigned> error_pos;
  crv_variable<unsigned> transmit_delay;
  ...

  crv_constraint c_read_write {inside(read_write(),
    std::set<ubus_rw_enum> {
      ubus_rw_enum::READ, ubus_rw_enum::WRITE
    })};

  crv_constraint c_size {inside(size(),
    std::set<int> { 1, 2, 4, 8 }
  )};

  crv_constraint c_data_wait_size {
    data().size() == size(),
    wait_state().size() == size()
  };
```

SystemC

Scope of Standardization

1. Constrained randomization of SystemC datatypes (C++)
2. Randomization of SC objects (i.e. classes) using standard datatypes
3. Constrained randomization of UVM-SystemC sequence items
4. Constrained randomization of UVM-SystemC sequences

Randomization using CRAVE – Example



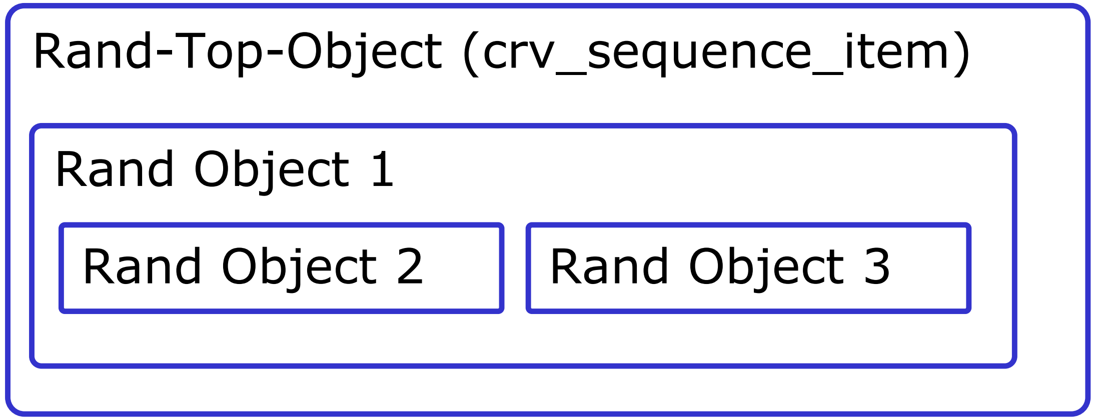
```
struct sysc_cont : public crv_sequence_item {
  crv_variable<sc_int  <5>>   x{ "x" };
  crv_variable<sc_uint<6>>   y{ "y" };
  crv_variable<sc_bv   <7>>   z{ "z" };

  sc_uint<5> t = 13;

  crv_constraint constr{ "constr" };

  sysc_cont(crv_object_name) {
   constr = { dist(x(), make_distribution(range<int>(5, 8))), y() > 0, y() % reference(t) == 0, y() != y(prev),
             (z() & 0xF) == 0xE };
  }
};
```

Special operators

- inside
- dist
- if_then
- if_then_else
- foreach
- unique
- bitslice

Randomization of SC objects using standard datatypes

- CRAVE builds a hierarchy of randomizable objects to randomize members



```
class rand_object: public crv_sequence_item {
public:
 crv_variable<unsigned> data;
 crv_constraint c_high_data { 50 < data(), data() < 100 };
 rand_object(crv_object_name) {
 }
};
```

- crv_sequence_item base class contains randomize() function
- Call to top object randomize, randomizes all objects below

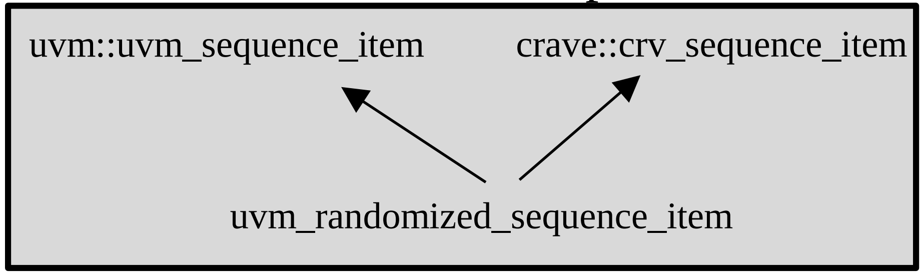
Randomization of SC objects using standard datatypes

- **Integration option 1 [considered difficult]**
  - Derive all sc_objects/uvm_object objects also from crv_sequence_item
  - User can randomize all objects, modules…
  - Very similar to SystemVerilog approach

  → High impact on current implementation
  → We would need support from SystemC LWG

- **Integration option 2 [current status]**
  - Derive just uvm_sequence_item from crv_sequence_item
  - Currently done for UBus (see presented example)

Multiple inheritance



Randomization of SC objects using standard datatypes

- **Integration option 3**
  - Keep CRAVE objects separate from SC/UVM objects
  - Initialize crv_variables with reference to original data member

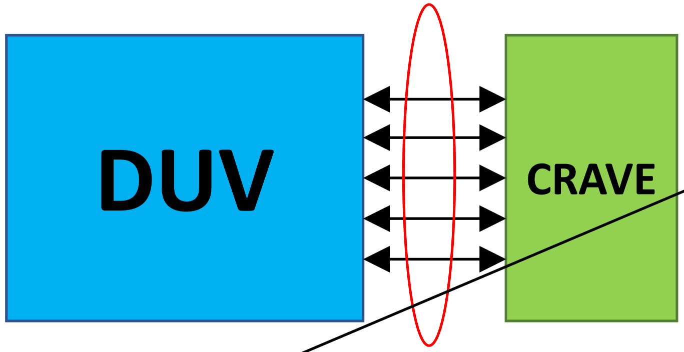
```
class rand_object: public crv_sequence_item {
public:
 crv_variable<unsigned> data;
 rand_object(crv_object_name nm, unsigned& data) :
  data(data){}
};
```

```
rand_object r("r", data);
r.randomize();
//access randomized members
if (data…)
```

  - Pro: Low impact on existing libraries → no changes to SystemC
  - Pro: No multiple inheritance
  - Pro: Randomize existing objects modules without changing original code (data members have to be public/accessible)
  - Con: User has to manage random objects more explicit

Integration Option 3 - Example

Pass by reference
Interface



```
Foo bar;
rand_object r("r", bar.data, bar.addr);
r.randomize();
//access randomized members
if ((bar.data…) && (bar.addr…))
```

```
class Foo {
public:
 unsigned data;
 unsigned addr;
};
```

```
class rand_object: public crv_sequence_item {
public:
 crv_variable<unsigned> data;
 crv_variable<unsigned> addr;
 rand_object(crv_object_name nm, unsigned& data, unsigned &addr) :
  data(data), addrs(addr) {}
};
```

CRAVE BUNDLE DEPENDENCIES

External Dependencies (Build infrastructure)

- Zlib development library, e.g. zlib1g-dev
- bzip2 development library, e.g. libbz2-dev
- CMake (at least v2.8.9)
- GCC (4.8.5)
- Git (download libraries)

External Dependencies - Libraries (not included in Bundle, will be downloaded automatically)

- Libraries
  - Boost (1.50.0) [Boost Software License][1]
    - Used for user constraint expressions (operators, conditions, distributions etc)
  - Glog (0.3.3) [BSD Licenses]
    - optional, if not available, then uses basic logging
    - **Example:** ***LOG***(*INFO*) << *"Partition has unsatisfiable hard constraints:";*
  - SystemC (2.3.1+) [Apache 2.0]
  - Better Enums used for CRAVE Enums [BSD 2-Clause "Simplified" License][2]

[1] It is a permissive license in the style of the BSD license and the MIT license, but without requiring attribution for redistribution in binary form.

[2] https://github.com/aantron/better-enums/blob/master/LICENSE.md

External Dependencies – Solvers (not included in Bundle, will be downloaded automatically)

- Solvers
  - Default build configuration
    - CUDD (version 3.0.0) [BSD 3-Clause License]
    - Z3 (version 4.6.0) [MIT License]
  - Optional Solvers
    - CVC4 (1.6) [BSD License]
    - Boolector (2.2.0) [restricted license for non-commercial use]
      - Lingeling SAT solver [restricted license for non-commercial use]
      - New version starting from 3.0.0 is MIT license (not integrated yet)
    - Yices2 (2.5.1) [GPLv3]
    - STP [MIT License]

Planned Configuration Script for CRAVE

- Configuration script for
  - Automatically looks for *SYSTEMC_HOME* environment variable
  - *-with-systemc* <path>
    - if SystemC not found downloads automatically

CRAVE INTEGRATION OPTIONS

CRAVE integration into POC (discuss with VWG)

- CRAVE included with UVM-SC-POC
  - UVM-SystemC/
    - Examples/
    - Src/
    - …
    - **CRAVE/**

- CRAVE external directory
  - UVM-SystemC/
    - Examples/
    - Src/
    - …
  - **CRAVE/**

Configuration Script for UVM-SystemC proof of concept implementation (CRAVE external)

- Configuration script for UVM-SystemC
  - Automatically looks for *SYSTEMC_HOME* environment variable
  - Automatically looks for *CRAVE_HOME* environment variable
  - *-with-systemc* <path>
  - *-with-crave* <path>
    - if CRAVE not found download manually or build without CRAVE support (discuss)

Next Steps

- Provide draft class definition in mark down language
  - For CRAVE variable (is independent from implementation option)
- Support by VWG
  - Test of current bundle by VWG
  - Joined git repository
  - Decision for integration option

Acknowledgments

Contributors

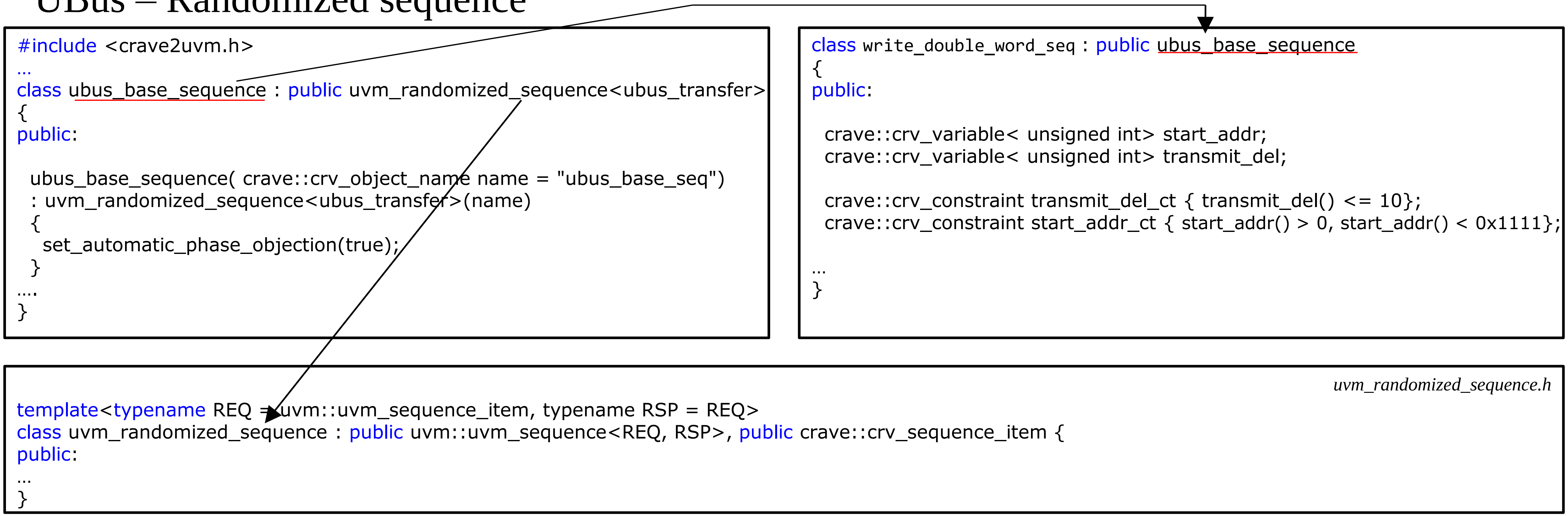
- Niklas Bruns
- Rolf Drechsler
- Tino Flenker
- Daniel Große
- Finn Haedicke
- Muhammad Hassan
- Hoang M. Le
- Dan Sörgel
- Thilo Vörtler
- Fereshta Yazdani

CRAVE has been supported in parts by the German Federal Ministry of Education and Research (BMBF) within the project CONVERS under contract no. 16ES0656, CONFIRM under contract no. 16ES0565, EffektiV under contract no. 01IS13022E and University of Bremens graduate school SyDe.

BACKUP

UBus – Randomized sequence

UBus – Randomized sequence



```
#include <crave2uvm.h>
…
class ubus_base_sequence : public uvm_randomized_sequence<ubus_transfer>
{
public:

  ubus_base_sequence( crave::crv_object_name name = "ubus_base_seq")
  : uvm_randomized_sequence<ubus_transfer>(name)
  {
    set_automatic_phase_objection(true);
  }
….
}
```

```
class write_double_word_seq : public ubus_base_sequence
{
public:

  crave::crv_variable< unsigned int> start_addr;
  crave::crv_variable< unsigned int> transmit_del;

  crave::crv_constraint transmit_del_ct { transmit_del() <= 10};
  crave::crv_constraint start_addr_ct { start_addr() > 0, start_addr() < 0x1111};

…
}
```

uvm_randomized_sequence.h

```
template<typename REQ = uvm::uvm_sequence_item, typename RSP = REQ>
class uvm_randomized_sequence : public uvm::uvm_sequence<REQ, RSP>, public crave::crv_sequence_item {
public:
…
}
```

UBus – Randomized sequence

```
class write_double_word_seq extends ubus_base_sequence;
  ...
  rand bit [15:0] start_addr;
  rand bit [7:0] data0; rand bit [7:0] data1; rand bit [7:0] data2;
  rand bit [7:0] data3; rand bit [7:0] data4; rand bit [7:0] data5;
  rand bit [7:0] data6; rand bit [7:0] data7;
  rand int unsigned transmit_del = 0;
  constraint transmit_del_ct { (transmit_del <= 10); }

  virtual task body();
  ...
```

SystemVerilog

```
template<typename REQ = ubus_transfer, typename RSP = REQ>
class write_double_word_seq : public ubus_base_sequence<REQ, RSP> {
public:
    crv_variable<sc_bv<16>> start_addr;
    crv_variable<sc_bv<8>> data0, data1, ... data7;
    crv_variable<unsigned int> transmit_del;
    crv_constraint transmit_del_ct { transmit_del() <= 10 };

    void body() {
    ...
```

SystemC

File: examples/ubus/vip/ ubus_master_seq_lib.h

UBus – Randomized sequence

```
`uvm_do_with(req,
     { req.addr == start_addr; req.read_write == WRITE; req.size == 8;
       req.data[0] == data0;
       req.data[1] == data1;
       ...
       req.data[7] == data7;
       req.error_pos == 1000;
       req.transmit_delay == transmit_del; }
)
```

SystemVerilog

```
UVM_DO_WITH (req,
        (req.addr() == start_addr(),
        req.read_write() == ubus_read_write_enum::WRITE,
        req.size() == 8,
        req.data()[0] == data0(),
        req.data()[1] == data1(),
        ...
        req.data()[7] == data7(),
        req.error_pos() == 1000,
        req.transmit_delay() == transmit_del())
);
```

SystemC

File: examples/ubus/vip/ubus_master_seq_lib.h

For simplicity only
One item here

Archive Contents of crave2uvm_<…> (1)

- Root dir in the following starting from contrib/crave/
- **CRAVE Layer:** src/include/
  - **uvm_randomized_sequence_item.h:**
    Extend UVM item w CRAVE functionality via multiple inheritance from *uvm_sequence_item* and *crv_sequence_item*
  - **uvm_randomized_sequence.h:**
    Extend UVM sequence w CRAVE functionality via multiple inheritance from *uvm_sequence* and *crv_sequence_item*
  - **crave2uvm.h:**
    UVM Macros, e.g. UVM_DO, …

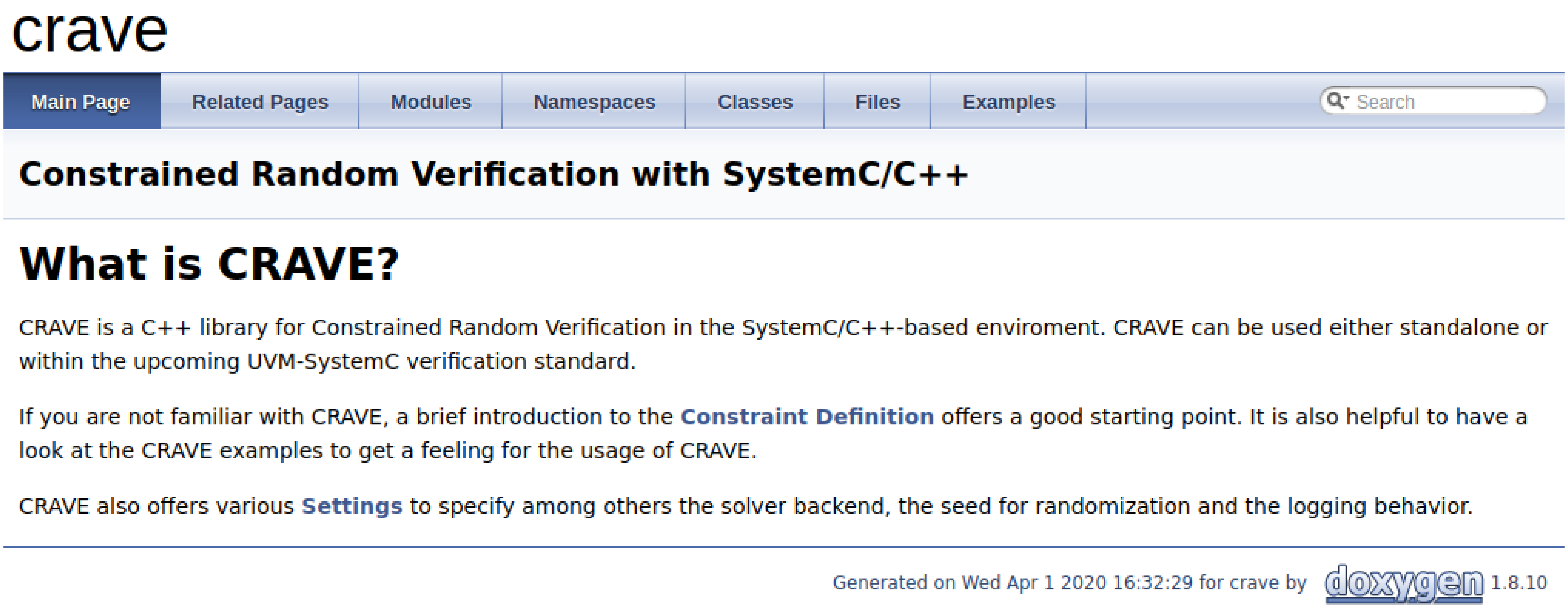
Archive Contents of crave2uvm_<…> (2)

- **CRAVE** (used by CRAVE layer): crave/src/crave/experimental/
  - Side note: "experimental" is the C++11-based CRAVE API
  - **SequenceItem.hpp:**
    Defines crv_sequence_item
  - **Expression.hpp:**
    Defines constraints expressions
  - …

doxygen (1)

- Root dir again contrib/crave/
- doxygen generation
  1. cd crave/
  2. make doxygen
- Accessing doxygen files
  1. cd doc/crave-doxygen/html
  2. View modules.html

doxygen (2)



Constrained Random Verification with SystemC/C++

What is CRAVE?

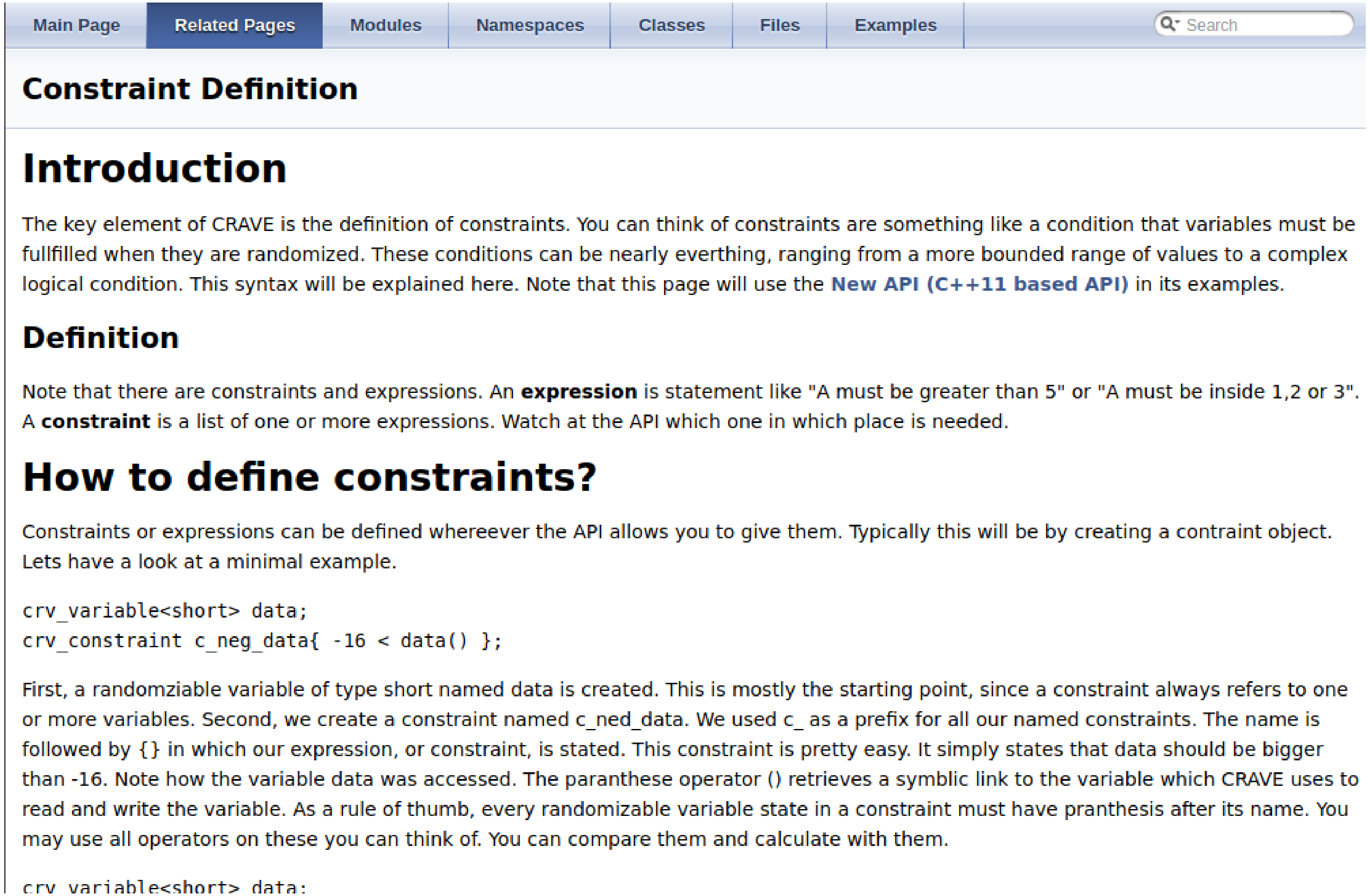
CRAVE is a C++ library for Constrained Random Verification in the SystemC/C++-based enviroment. CRAVE can be used either standalone or within the upcoming UVM-SystemC verification standard.

If you are not familiar with CRAVE, a brief introduction to the **Constraint Definition** offers a good starting point. It is also helpful to have a look at the CRAVE examples to get a feeling for the usage of CRAVE.

CRAVE also offers various **Settings** to specify among others the solver backend, the seed for randomization and the logging behavior.

Generated on Wed Apr 1 2020 16:32:29 for crave by doxygen 1.8.10

doxygen (3)



Constraint Definition

Introduction

The key element of CRAVE is the definition of constraints. You can think of constraints are something like a condition that variables must be fullfilled when they are randomized. These conditions can be nearly everthing, ranging from a more bounded range of values to a complex logical condition. This syntax will be explained here. Note that this page will use the **New API (C++11 based API)** in its examples.

Definition

Note that there are constraints and expressions. An **expression** is statement like "A must be greater than 5" or "A must be inside 1,2 or 3". A **constraint** is a list of one or more expressions. Watch at the API which one in which place is needed.

How to define constraints?

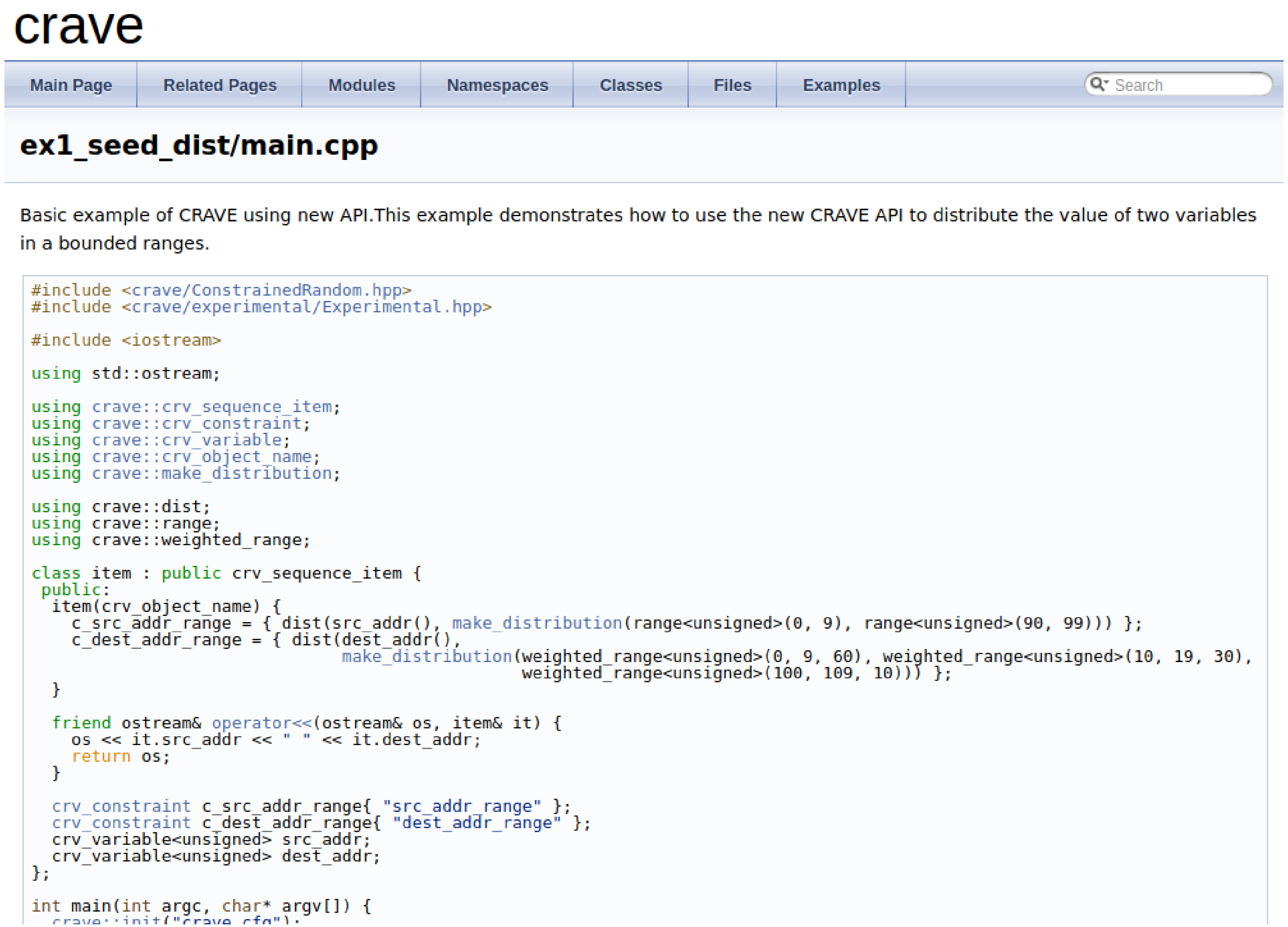
Constraints or expressions can be defined whereever the API allows you to give them. Typically this will be by creating a contraint object. Lets have a look at a minimal example.

```
crv_variable<short> data;
crv_constraint c_neg_data{ -16 < data() };
```

First, a randomziable variable of type short named data is created. This is mostly the starting point, since a constraint always refers to one or more variables. Second, we create a constraint named c_ned_data. We used c_ as a prefix for all our named constraints. The name is followed by {} in which our expression, or constraint, is stated. This constraint is pretty easy. It simply states that data should be bigger than -16. Note how the variable data was accessed. The paranthese operator () retrieves a symblic link to the variable which CRAVE uses to read and write the variable. As a rule of thumb, every randomizable variable state in a constraint must have pranthesis after its name. You may use all operators on these you can think of. You can compare them and calculate with them.

```
crv_variable<short> data;
```

doxygen (4)



crave

ex1_seed_dist/main.cpp

Basic example of CRAVE using new API.This example demonstrates how to use the new CRAVE API to distribute the value of two variables in a bounded ranges.

```
#include <crave/ConstrainedRandom.hpp>
#include <crave/experimental/Experimental.hpp>

#include <iostream>

using std::ostream;

using crave::crv_sequence_item;
using crave::crv_constraint;
using crave::crv_variable;
using crave::crv_object_name;
using crave::make_distribution;

using crave::dist;
using crave::range;
using crave::weighted_range;

class item : public crv_sequence_item {
 public:
  item(crv_object_name) {
    c_src_addr_range = { dist(src_addr(), make_distribution(range<unsigned>(0, 9), range<unsigned>(90, 99))) };
    c_dest_addr_range = { dist(dest_addr(),
                               make_distribution(weighted_range<unsigned>(0, 9, 60), weighted_range<unsigned>(10, 19, 30),
                                                 weighted_range<unsigned>(100, 109, 10))) };
  }

  friend ostream& operator<<(ostream& os, item& it) {
    os << it.src_addr << " " << it.dest_addr;
    return os;
  }

  crv_constraint c_src_addr_range{ "src_addr_range" };
  crv_constraint c_dest_addr_range{ "dest_addr_range" };
  crv_variable<unsigned> src_addr;
  crv_variable<unsigned> dest_addr;
};

int main(int argc, char* argv[]) {
  crave::init("crave.cfg");
```