

HelenOS Architecture Description

Martin Děcký

DISTRIBUTED SYSTEMS RESEARCH GROUP

<http://dsrg.mff.cuni.cz/>

**CHARLES UNIVERSITY IN PRAGUE
FACULTY OF MATHEMATICS AND PHYSICS**



Heads-up: HelenOS feature set

- Prerequisite for a reasonable OS research
 - An operating system we are familiar with
 - A good balance of real-life features and complexity
 - Linux is way too complex and unstructured
 - Some (of us) have serious concerns about some design and implementation decisions in the *L4 family
 - Synchronous IPC
 - Unportable code cluttered with low-level hacks
 - Great existing and new emerging operating systems we try to take inspiration from
 - Plan9/Inferno, QNX, Spring, K42, Spring, JX/JNode
 - MINIX3, Singularity/Midori, Coyotos, seL4, Barrelfish



Heads-up: New in HelenOS

- Almost feature-complete microkernel
 - Not a fundamentalistic microkernel
 - Sustained ports for 7 hardware architectures (3 more are underway)
 - Only one major redesign under scope
 - Better use of on-demand mapping in kernel address space
 - Only one major hardware feature not supported yet
 - Arbitration of DMA/bus mastering
- User space features
 - User space device drivers, devices management
 - Block devices subsystem
 - RAM disk, IDE/ATA-6/ATAPI, SATA underway
 - Generic VFS subsystem
 - File system drivers for FAT, tmpfs, DevFS, UDF underway
 - Interactive shell, I/O redirection, pipes underway



Heads-up: New in HelenOS (2)

- Full Unicode Character Set (UCS) support
- Debugging and tracing framework
 - Dynamic linking underway
- *Almost there*
 - Networking underway
 - Task snapshoting underway
 - Hierarchical device drivers interface underway
 - Support for MMU-less architectures
 - Caching infrastructure underway
 - Access rights management
 - Real-time features
- **Main implementation goal: Self-hosting**
- **Main research goal: Formal verification of correctness**



Motivation: OS correctness

- *Correct operating system*
Bug-free operating system
 - Intuitively we see all the points and benefits
 - But *correctness* and *bug freedom* are not formal properties
 - Correct with respect to what?
 - Free from what bugs? What is a *bug*?
 - **(Formal) architecture description**
(Formal) behavior description
 - Means to base our reasoning on



Step back: Components?

- The components are still there
 - Actually, they are an important prerequisite
 - Primitive components as first-class entities in architecture description
 - Components' interfaces connected via bindings
 - Composite components as subsystems
 - Components as means of encapsulation
 - On the implementation level
 - On the behavior description level
 - A way how to check compliance with some properties in an isolated manner
 - Still just induced by the design (no framework/CAL)



HelenOS sources
C99 with GNU extensions



HelenOS sources

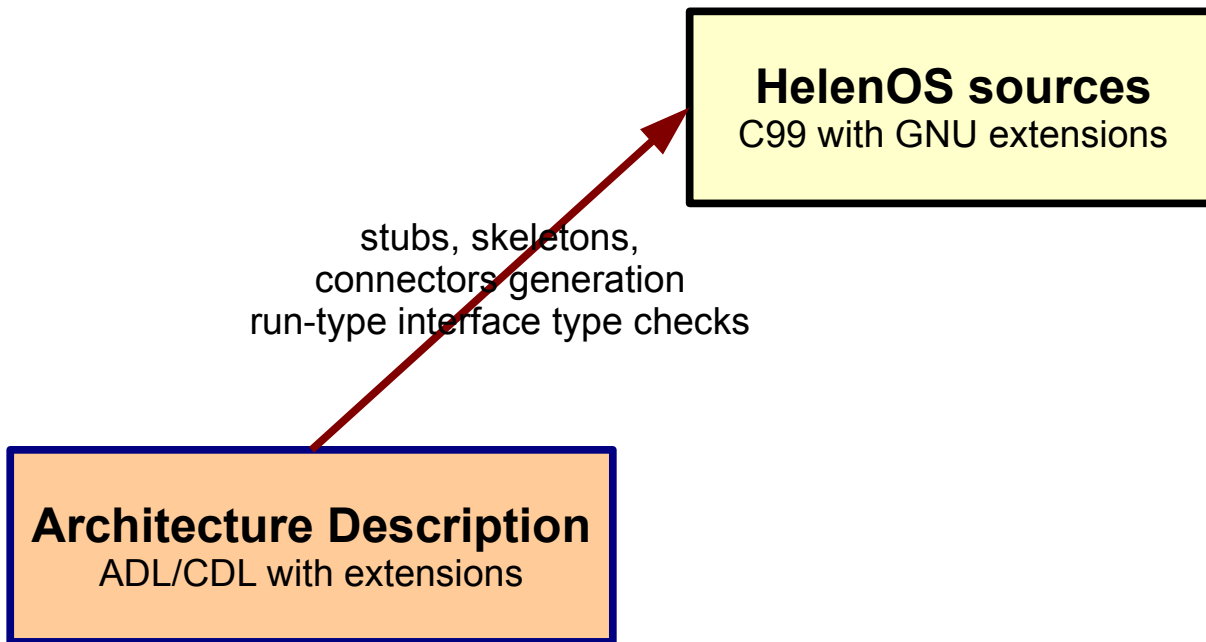
- Not much, but at least some checks
 - Implicit components and interfaces (by design)
 - C99 language features and modern compiler checks
 - Type checking
 - If you don't use `void *` and integer arrays for everything
 - Static analysis
 - Strict aliasing rules, missing/unreachable return values, missed enum values in switches, uninitialized/unused variables, missing function prototypes, unsafe sign/unsigned conversions, unreachable code
 - HelenOS is compiled with `-O3` and most of compiler warnings as fatal errors



HelenOS sources (2)

- GNU extensions
 - No-return functions, twice-return functions
 - Precising the computational model
- More static checking possible thanks to new emerging compilers (LLVM/Clang)
 - Clang Static Analyzer
 - Custom code analysis thanks to access to the full AST
 - Checking for various custom-defined bugs (adherence to coding standards)
- Abstract interpretation analysis
 - Stanford Checker/Coverity
 - Identification of common programming defects
 - Specific concurrency defects (race conditions, deadlocks), memory issues (leaks, null-dereference, use-after-free, double free)





Extended ADL/CDL

- Based on SOFA (1.x) ADL
 - Just a personal preference
 - IDL-ish language is more human-readable than XML
 - Constituents
 - Interfaces
 - Description of IPC method signatures
 - Input/output arguments passed by value
 - Type `ipcarg_t` (native integer for fast IPC messages)
 - Input/output arguments passed by memory copying
 - Type `stream` (passing arbitrary data)
 - Type `string` (passing null-terminated strings)
 - Memory sharing not captured in method signatures
 - The correct semantics requires to send and IPC message with every data passed via memory sharing



Extended ADL/CDL (2)

- Optional expected behavior protocol
 - Used for specifying correct sequencing of IPC methods
 - Especially useful for passing arguments via memory copying
- Interface inheritance
 - Support for basic specialization of interfaces, is-a relationship (e.g. “*is a service*”, “*is a device driver*”, “*is a block device driver*”)
- **Frames (primitive components)**
 - HelenOS uses flat physical component model
 - The behavior is only defined on primitive components
 - Provided & required interfaces
 - Initialization, life-time and finalization behavior protocol
 - Extends interface behavior protocols where we need to capture the interplay between the methods of various interfaces



Extended ADL/CDL (3)

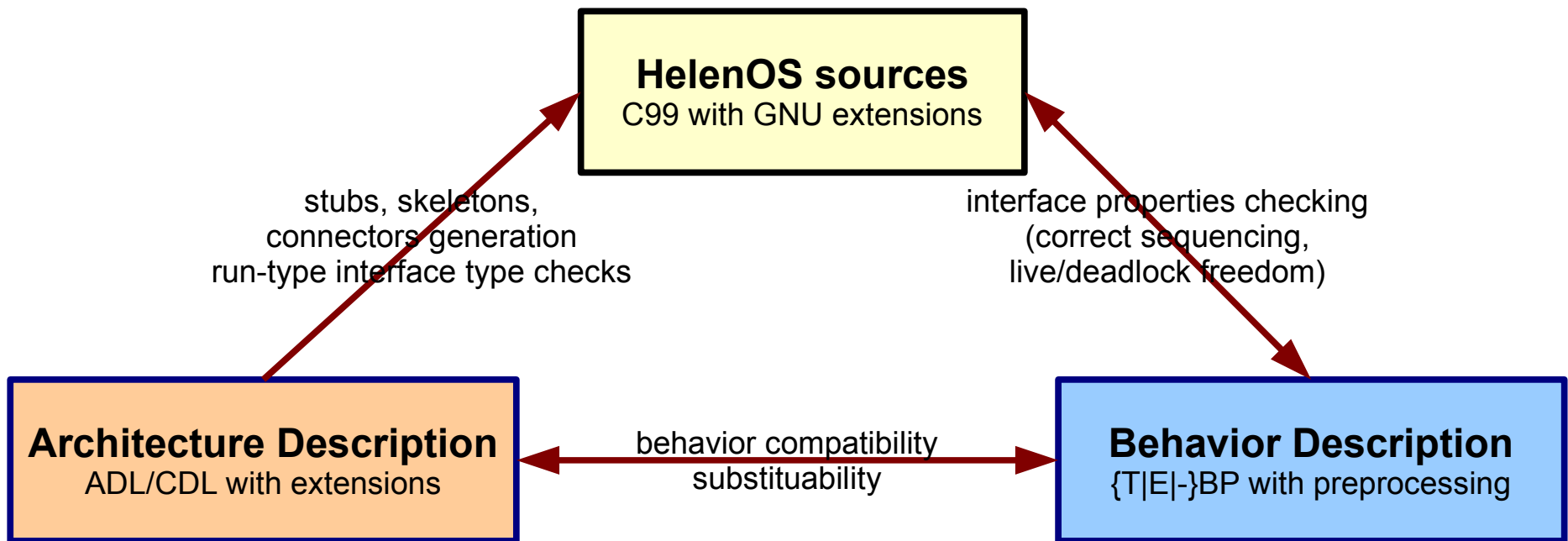
- Architectures (composite components)
 - Represent subsystems
 - Only a design feature so far, a composite component cannot define its own behavior
 - Captures interface bindings, delegations and subsumptions
- System architecture
 - Captures the top-level components and their bindings
- Simple preprocessing
 - Includes with substitution
 - Mirrors the use of libraries in C code



Extended ADL/CDL (4)

- Uses
 - Generating final set of $\{T|E|- \}$ BPs which are used as input for checkers
 - Drawing nice figures of the architecture
 - The aesthetical qualities need some improvement ...
 - Future use
 - Replacement of hand-written IPC stubs and skeletons
 - Possible generation of connectors/adapters (if we move forward to different message passing mechanisms)
 - Run-time interface identification and introspection
 - Replacement for the current “duck-typing” approach





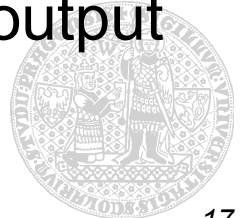
Behavior Protocols

- A common subset of core {T|E|-}BP
 - Only the core behavior description used (traces)
 - Method invocation, method response
 - Nested methods
 - Operators
 - Repetition
 - Sequence
 - Alternative
 - And-parallel
 - Or-parallel (currently unused)
 - Parentheses



Behavior Protocols (2)

- Other of EBP and TBP not used
 - Architecture description is derived from ADL
 - The `.archbp` file, instantiation of components, `component { }` clauses, etc. are generated automatically
 - Enum types, reactions, etc. omitted
 - They might be handy at a later point
- Preprocessing
 - Includes with substitution
 - Mirrors the repeated use of functions in C code
 - Two simple macros (`tentative`, `alternative`)
 - *hadlbp* preprocessor creates a BP, EBP or TBP output by merging interface protocols (including inherited protocols) and frame protocols



Behavior Protocols (3)

```
tentative {  
    !loader.ipc_m_connect_to_me  
}
```



```
(  
    !loader.ipc_m_connect_to_me +  
    NULL  
)
```

```
if (cond)  
    ipc_connect_to_me(phone, arg1, arg2, arg3, &hash);
```



Behavior Protocols (4)

```
?sync {  
    alternative (fs; tmpfs; fat; devfs) {  
        !fs.sync  
    }  
}
```



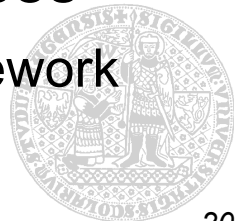
```
(  
    !tmpfs.sync +  
    !fat.sync +  
    !devfs.sync  
)
```

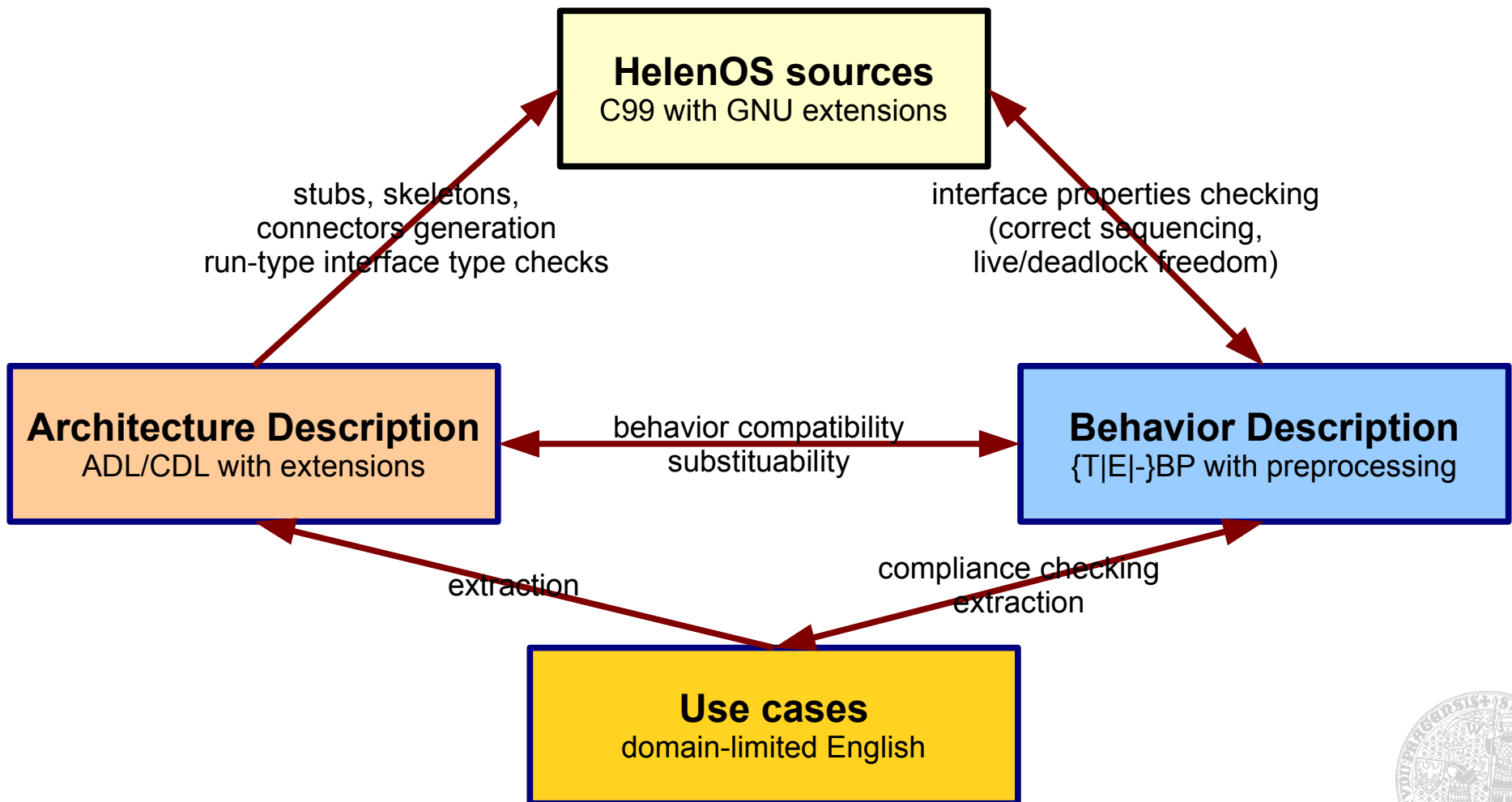
```
int fs_phone = vfs_grab_phone(file->node->fs_handle);  
  
ipc_call_t answer;  
aid_t msg = async_send_2(fs_phone, VFS_OUT_SYNC,  
    file->node->dev_handle, file->node->index, &answer);
```



Behavior Protocols (5)

- Uses
 - Currently the protocols are written as a specification
 - Checking for communication compatibility between components (bad activity, no activity)
 - Future use
 - Checking for compliance of the specification with the code (on the interface level)
 - Checking of substitutability of components
 - Run-time checking of communication correctness (with unknown components) and debugging purposes
 - Replacement for the simple protocols in the tracing framework

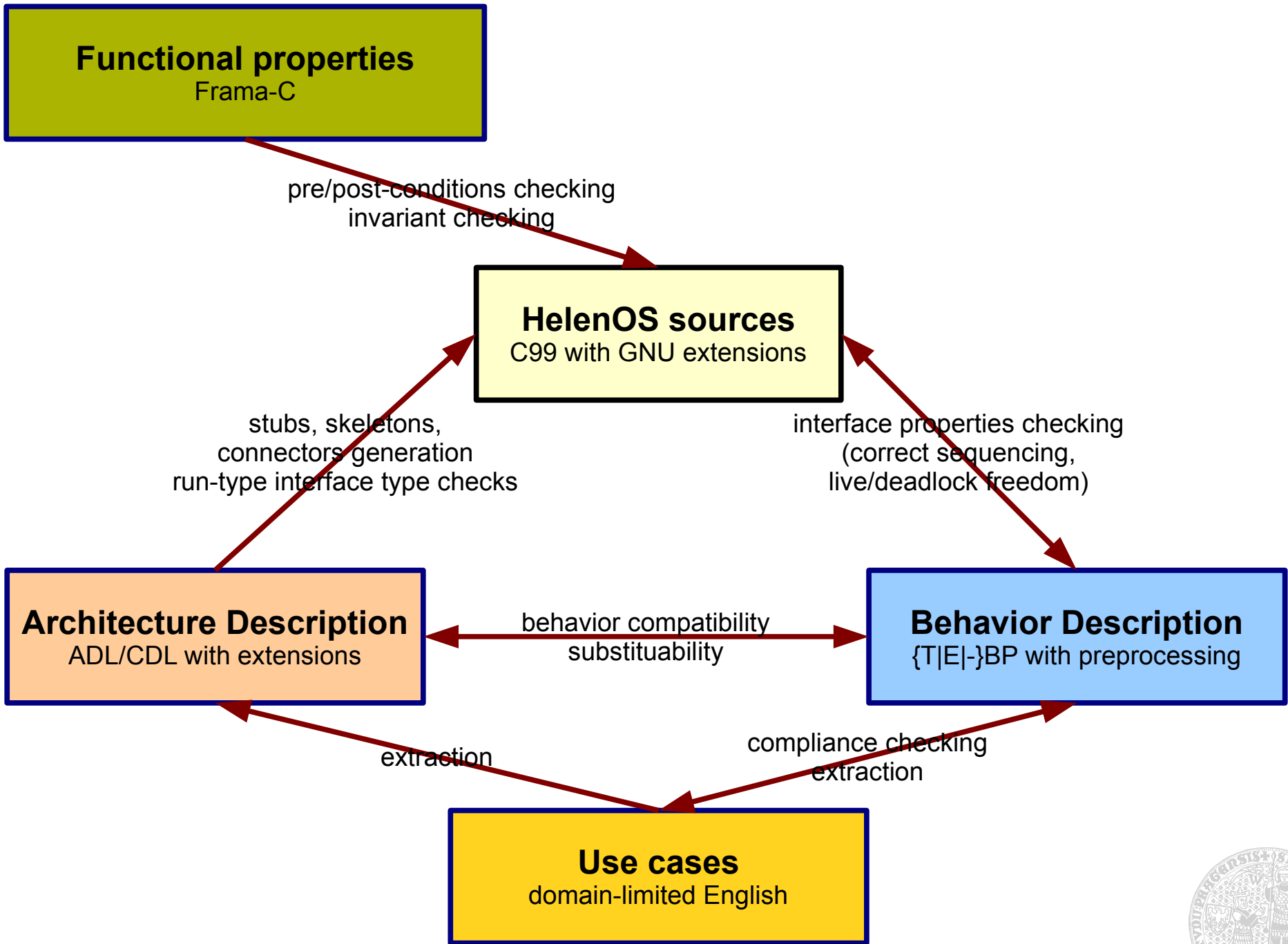




TBD: Use cases

- Automatic conversion of textual use cases to formal behavior description
 - *Procasor* is probably a good starting point
 - Not the most important direction to follow right now
 - But interesting possible side effects
 - Formal description of the domain
 - Formal description of primitive operations semantics
 - Formal description of the computational model





TBD: Functional properties

- Capturing correctness of the code within components
 - In terms of function/block pre/post-conditions, invariants
 - *Frama-C/why* is probably a good starting point
 - Detecting logical/semantical bugs beyond the reach of compilers and static code analyzers
 - Function contracts
 - Data structures integrity constrains
 - Locking order
 - Finer-grained race conditions



```

/** Allocate a file descriptor.
 *
 * @param desc If true, look for an available file
 *             descriptor in a descending order.
 *
 * @return First available file descriptor or a
 *         negative error code.
 */
int vfs_fd_alloc(bool desc)
{
    if (!vfs_files_init())
        return ENOMEM;

    unsigned int i;
    if (desc)
        i = MAX_OPEN_FILES;
    else
        i = 0;

    while (true) {
        if (!files[i]) {
            files[i] = (vfs_file_t *)
                malloc(sizeof(vfs_file_t));
            if (!files[i])
                return ENOMEM;

            memset(files[i], 0, sizeof(vfs_file_t));
            fibril_mutex_initialize(&files[i]->lock);
            vfs_file_addrf(files[i]);
            return (int) i;
        }
    }
}

```

```

        if (desc) {
            if (i == 0)
                break;

            i--;
        } else {
            if (i == MAX_OPEN_FILES)
                break;

            i++;
        }
    }

    return EMFILE;
}

```

uspace/srv/vfs/vfs_file.c



```

/** Allocate a file descriptor.
 *
 * @param desc If true, look for an available file
 *            descriptor in a descending order.
 *
 * @return First available file descriptor or a
 *         negative error code.
 */
int vfs_fd_alloc(bool desc)
{
    if (!vfs_files_init())
        return ENOMEM;

    unsigned int i;
    if (desc)
        i = MAX_OPEN_FILES;
    else
        i = 0;

    while (true) {
        if (!files[i]) {
            files[i] = (vfs_file_t *)
                malloc(sizeof(vfs_file_t));
            if (!files[i])
                return ENOMEM;

            memset(files[i], 0, sizeof(vfs_file_t));
            fibril_mutex_initialize(&files[i]->lock);
            vfs_file_addrf(files[i]);
            return (int) i;
        }
    }
}

```

```

    if (desc) {
        if (i == 0)
            break;

        i--;
    } else {
        if (i == MAX_OPEN_FILES)
            break;

        i++;
    }
}

return EMFILE;
}

```

uspace/srv/vfs/vfs_file.c



```

/** Unregister task from IRQ notification.
 *
 * @param box Answerbox associated with the notification.
 * @param inr IRQ number.
 * @param devno Device number.
 *
 * @return EOK on success, ENOENT on unregistered IRQ.
 */
int ipc_irq_unregister(answerbox_t *box, inr_t inr, devno_t devno)
{
    /* ... */

    irq = hash_table_get_instance(lnk, irq_t, link);

    /* ... */

    /*
     * We need to drop the IRQ lock now because hash_table_remove() will try
     * to reacquire it. That basically violates the natural locking order,
     * but a deadlock in hash_table_remove() is prevented by the fact that
     * we already held the IRQ lock and didn't drop the hash table lock in
     * the meantime.
     */
    spinlock_unlock(&irq->lock);

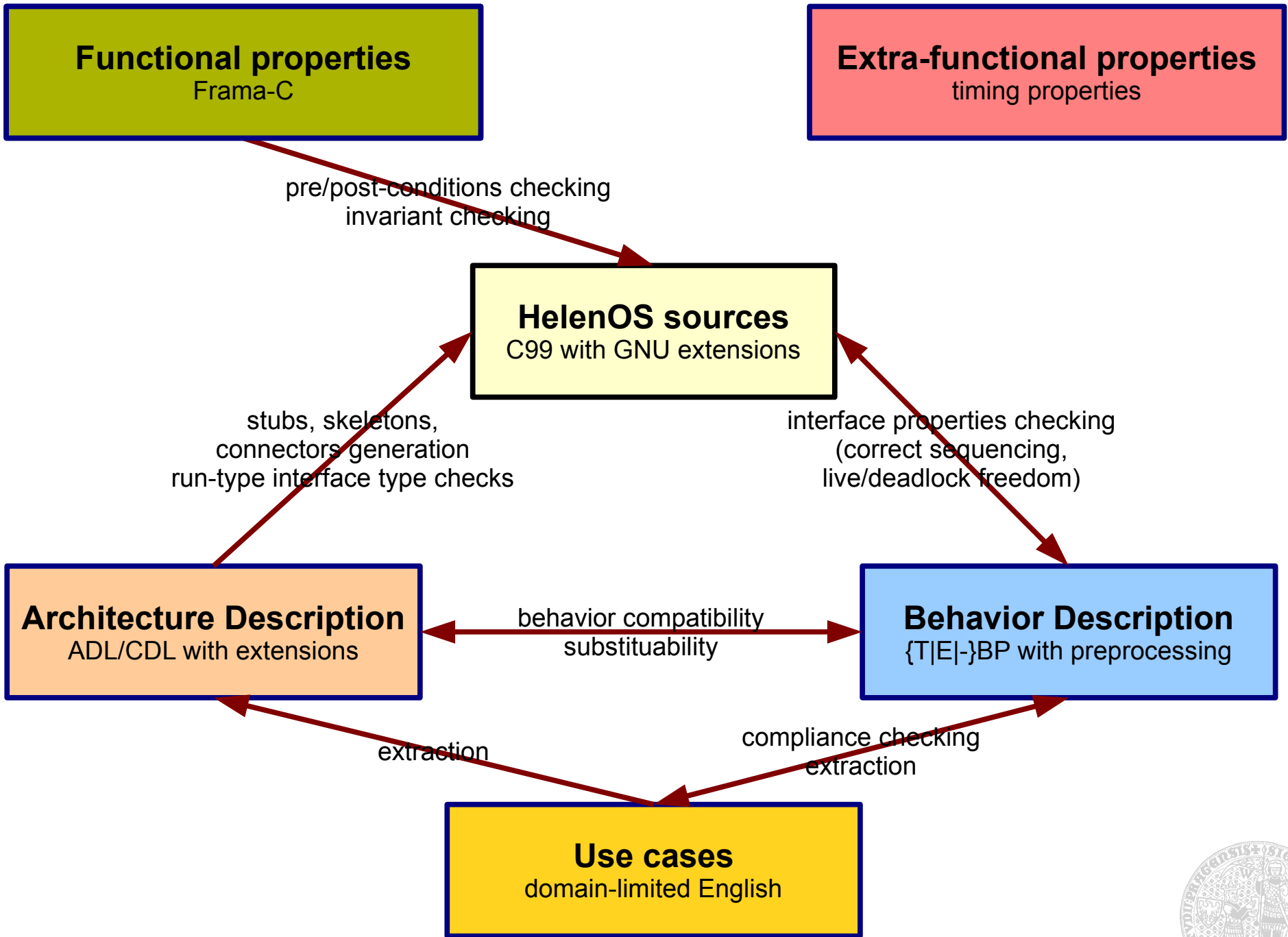
    /* ... */

    return EOK;
}

```

kernel/generic/src/ipc/irq.c





Challenges

- Kernel
 - Still just a single component
 - Too many complex dependencies
 - Complex locking schemes
 - Typical kinds of bugs
 - Deadlocks
 - Memory leaks
 - Buffer overflows



Challenges (2)

- User space
 - Fine-grained components
 - But no interfaces, method signatures, etc. at run-time so far (duck-typing)
 - Dynamic architecture
 - Preset bindings created at run-time (*Naming service*)
 - The ADL/BP describes a snapshot of the architecture
 - Configuration and/or the user induce more bindings (*Device mapper*)
 - File systems can be mounted/unmounted
 - File system drivers can have multiple instances
 - Each end user application can connect to various devices



Challenges (3)

- Multiple instances, self-references
- Unusual threading model
 - Cooperatively scheduled fibrils (optionally combined with common threads)
 - For BP: not a serious issue (given correct message serialization)
 - We need means to check the correctness of the message serialization itself
- Performance-optimized IPC
 - Memory sharing
 - Message forwarding
 - In BP: emitting a new message
 - Creating and closing connections, callback connections
 - In BP: hard to express, dynamic nature of the architecture



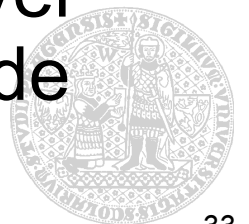
Current progress and issues

- Architecture and behavior of HelenOS
 - Fully sketched in ADL and “BP”
 - The protocols capture all intentions, but are not correct
 - Starting from an empty set and slowly adding protocols and features one-by-one
 - Issues with reentrancy
 - Each component's interface has a reentrancy count equal to the multiplicity of the incoming binding
 - Reentrancy is provided by multiplying interface protocols and joining the copies using or-parallel operator
 - Currently 13 component protocols with the highest reentrancy count of 4 → state space too large to be processed with *bp2promela* (with 2 GB of RAM)



Open questions

- Trace semantics and its suitability for real-life applications
 - **Nightmare scenario:** A plain sequence of calls, but after each one the return value is checked for error
 - Somehow related to Vilda's proposed BP extension for exceptions
- Protocols of common functions and libraries
- Level of abstraction in protocols
 - **Nightmare scenario:** Non-trivial file system driver with a block cache → hundreds of possible code paths resulting in different IPC method traces



Reference and disclaimer

`bzr://bzr.helenos.org/mainline`

The work on ADL, BP and other related topics wouldn't be possible without the previous work of the component guys, the formal guys and all the other members of DSRG

Special thanks for extremely helpful consultations go to Jan Kofroň, Ondřej Šerý and Petr Tůma

Thanks goes also to all the current and past contributors to the HelenOS project

