Time and Space Partitioning kernel formalisation

Andrew Butterfield, David Sanán
Lero @ Trinity College Dublin

Time And Space Partitioning Kernel Formalisation: Objectives

- Savoir-IMA is exploring a time-space partitioning (TSP) kernel
- This part of MTOBSE looked at:
  - developing a Reference Specification
  - formalising the specification (Abstract Specification)
  - exploring feasibility/costs of formal verification.
- High level goal:
  - move towards Common-Criteria/Separation Kernel Protection Profile certification standards (EAL5+)

Time And Space Partitioning Kernel Formalisation: Deliverables

Key Deliverables:

- *Reference* Specification for a partitioning and separation kernel.
- Formal *Abstract* Specification of the above
- Formal Methods Toolset

Support Material:

- Formalism Evaluation and Choice
- Abstract Specification Documentation
- Formal Verification Process Assessment

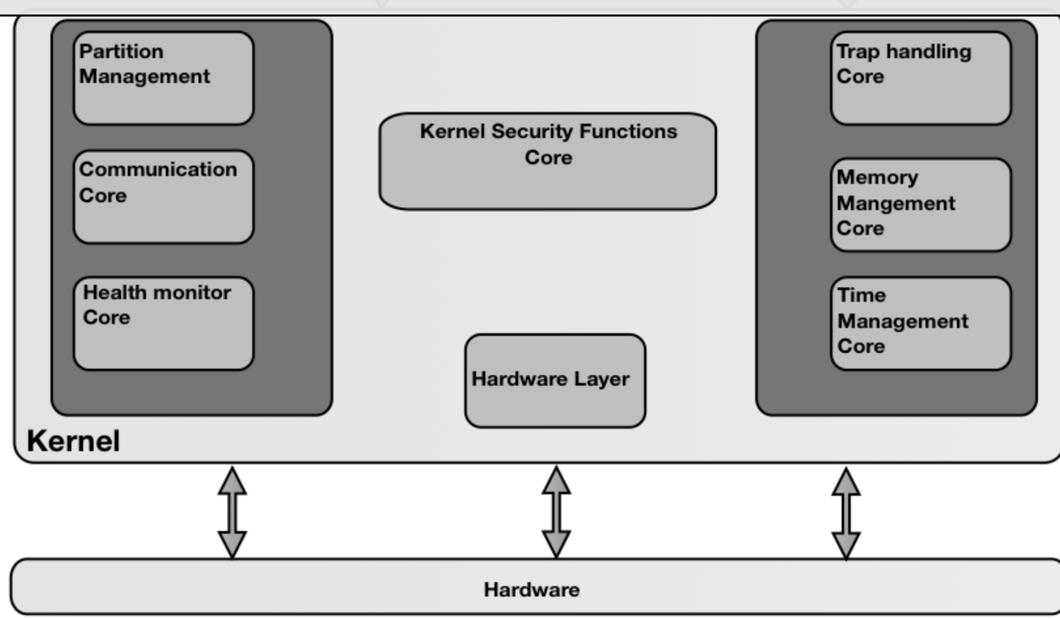
Reference Specification

- Reference Specification comprises (ECSS-E-ST-40C):
  - Software Requirement Specification (SRS)
  - Interface Control Document (ICD)
  - Architecture Design Document (ADD)
- Key Inputs:
  - Pre-existing specifications: AIR-2, XtratuM, PikeOS, sel4.
  - Common Criteria (CC)
  - Separation Kernel Protection Profile (SKPP)
  - IMA-SP SEP SSS
  - IMA-SP TSP Services Specification
  - ARINC 653

Software Requirements

The design is based in three majors logical blocks:

- Partitioning Functionality.
- Low level Functionality.
- Security Functionality.



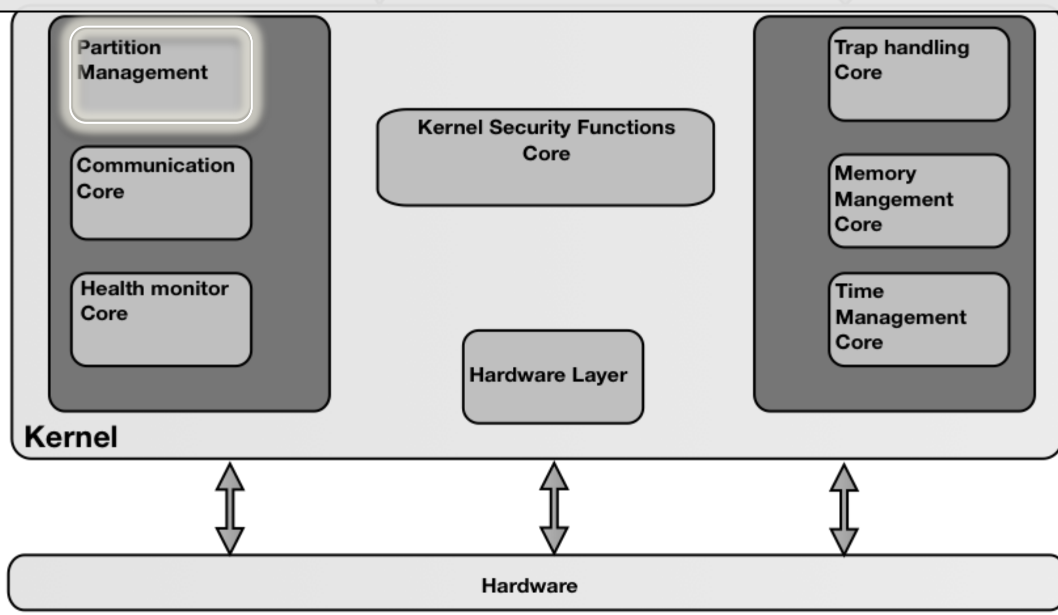
Software Requirements

What is the configuration data of a partition?

What are the states of a partition?

How are resources assigned to partitions?

How are partition scheduled?

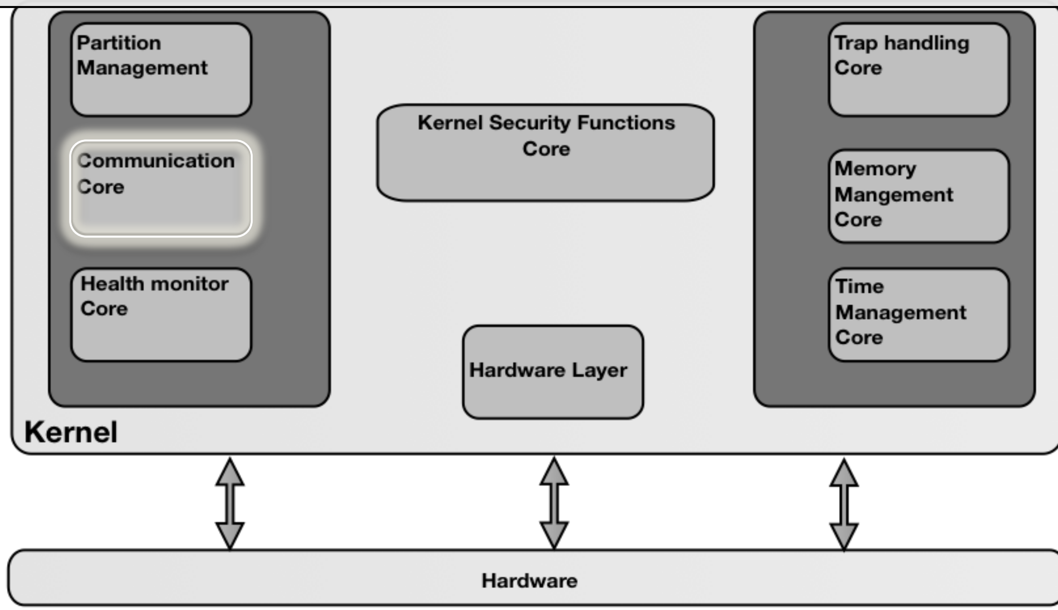


Software Requirements

What mechanisms do partitions do for communication?

What types of communication between partitions are possible?

What are the characteristics of the communication?



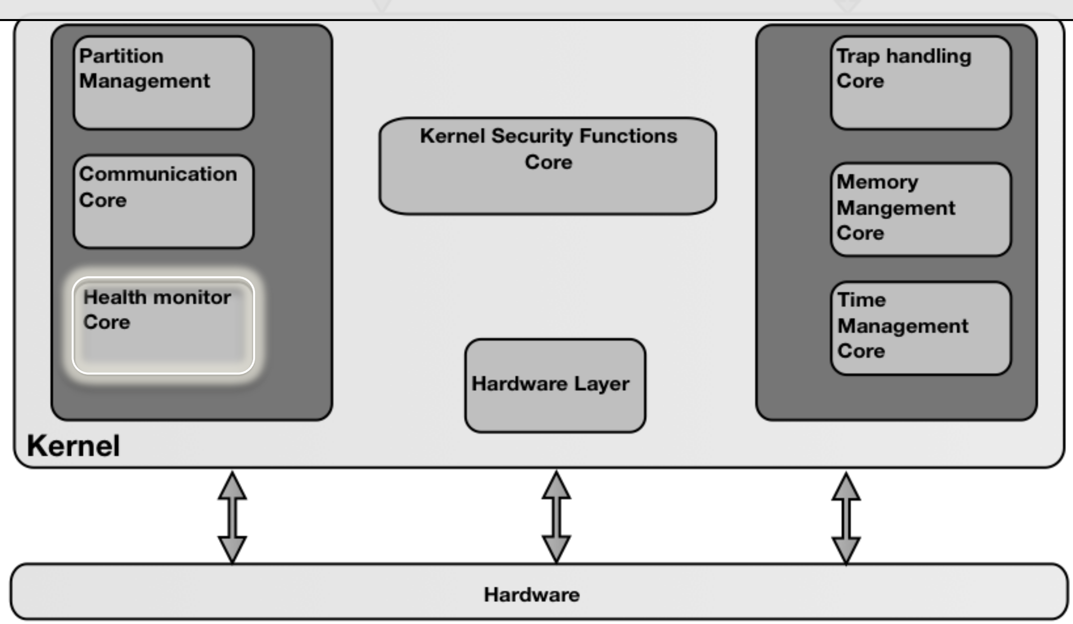
Software Requirements

How are faults managed?

How are events stored and who audits them?

What error levels are managed?

What to do on a partition fault?

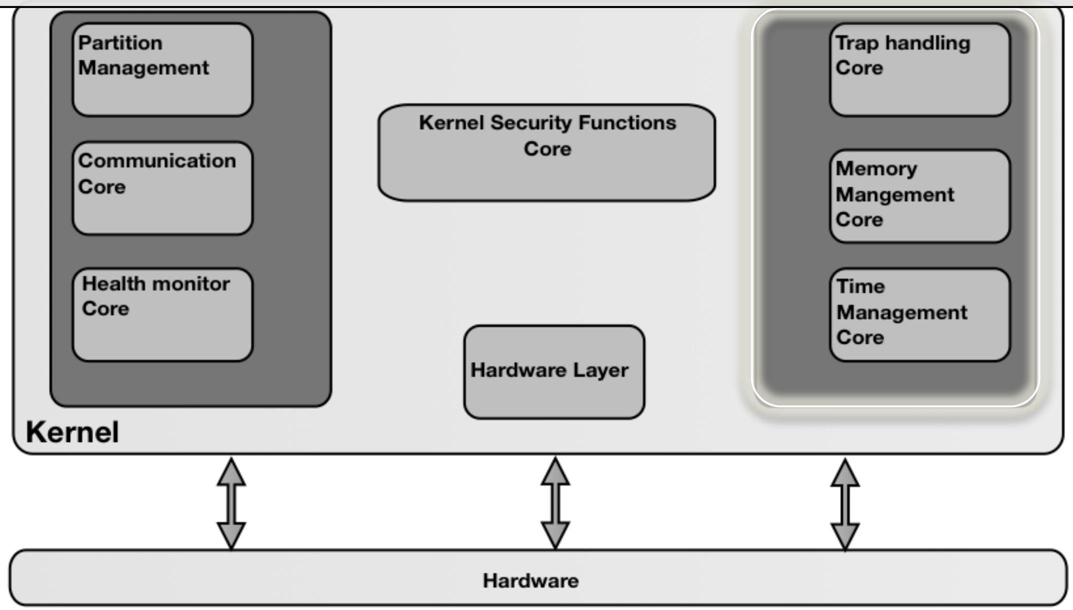


Software Requirements

How memory mechanisms are managed?

How traps are handled?

What clocks are needed?



Software Requirements

What kernel events must be audited?

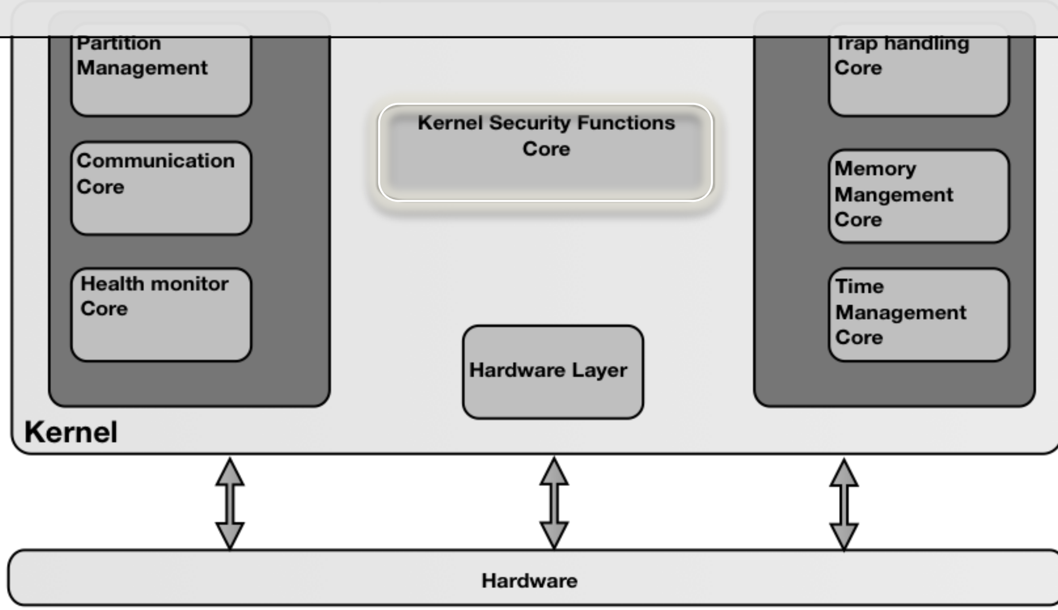
What flow between partitions is allowed?

How to handle flows not explicitly assigned?

What is the necessary partition data for the security?

What rules on that data must be enforced?

What privileges do partitions have?



Software Requirements

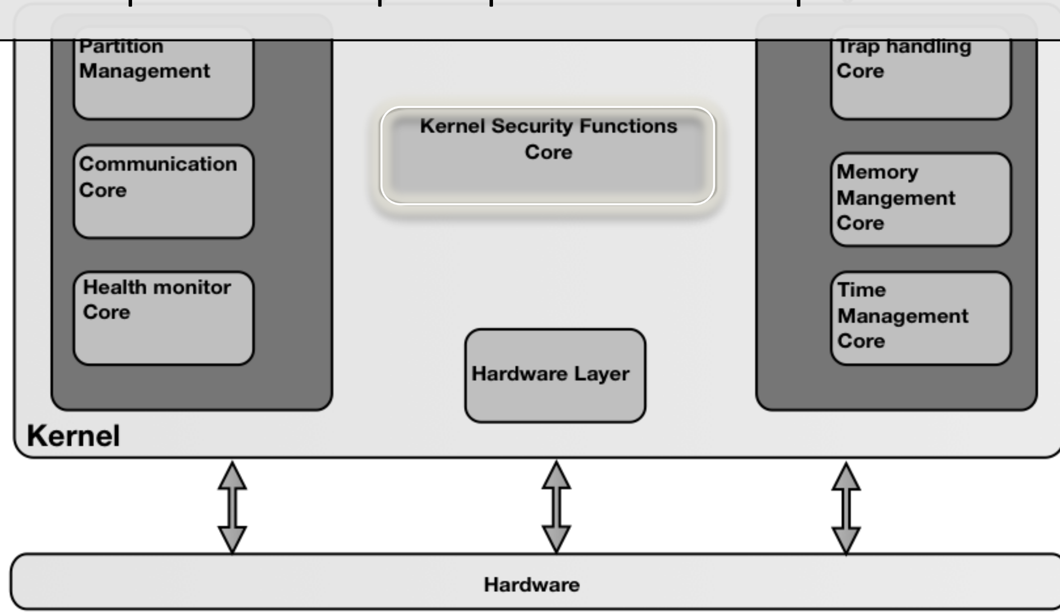
In what functional states can the kernel be? What operations can be performed on the kernel?

What is the policy to change kernel configuration?

What is a kernel secure state? How to detect it?

What to do when kernel failures are detected?

Control of spatial and temporal quotas allowed to partitions.



Interface Control

These logical components provides user and system partitions with services.
Based on IMA-SP, Arinc 653, and SKPP.

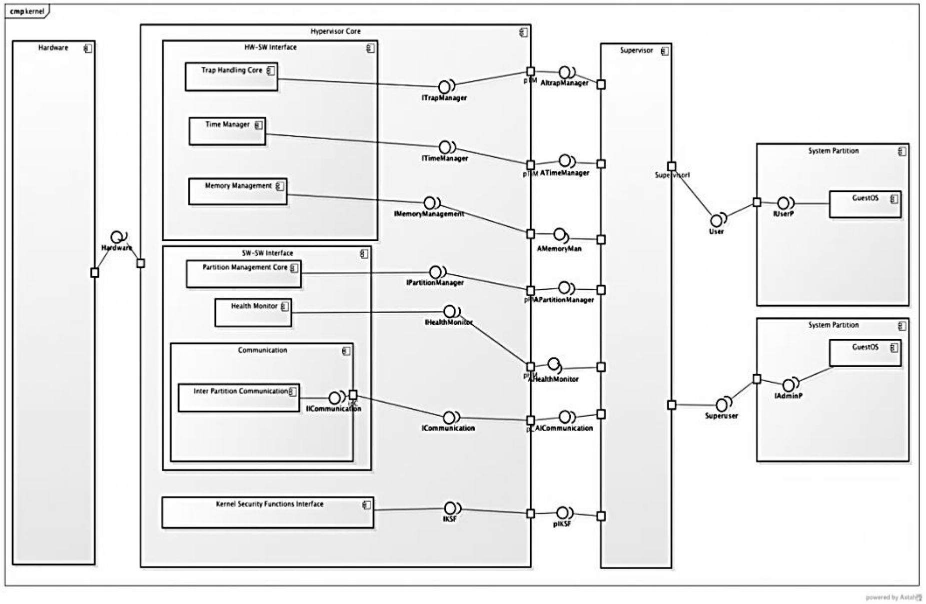
Partition:
- mode and status.

Communication:
- port creation, messages, config.

HealthMonitor:
- access monitoring logs.

Virtual Machine:
- clocks, timers, cache, interrupts…

Kernel:
- restart, halt, re-schedule…



From the specification to the implementation

Reference Specification

Requirement Partition Control (PM_PC)
From ARINC 653P1-3 2.3.1.2

The kernel shall control and maintain the current state of each partition by means of a partition control table storing the partition state. The partition control table maintains the following list of attributes for each partition:
Partition current state.
Partition initialization reason.

Requirement Partition Current State (PM_PCS)
From IMA-SP SEP SSS v.1.1 IMA_SEP_900, IMA_SEP_901, IMA_SEP_902, IMA_SEP_903, and IMA_SEP_904

Partitions can be in the following states:
Cold_Start - Warm_Start: Initialization of the partition. The kernel is only allowed to create intercommunication channels in the start states. Warm and Cold start are used to differentiate when it has been possible to retain data from before the restart. At restart of the system all partitions shall be in the Cold_Start state.
IDLE: Termination state of the partition.

…

?

C Code

```
#ifndef _XM_KTHREAD_H_
#define _XM_KTHREAD_H_

#include <assert.h>
#include <guest.h>
#include <ktimer.h>
#include <xmconf.h>
#include <xmef.h>
#include <objdir.h>
#include <arch/kthread.h>
#include <arch/atomic.h>
#include <arch/irqs.h>
#include <arch/xm_def.h>
#ifdef CONFIG_OBJ_STATUS_ACC
#include <objects/status.h>
#endif

#ifndef _XM_KERNEL_
#error Kernel file, do not include.
#endif

struct guest {
   struct xmcPartition *cfg;
   struct kThreadArch kArch;
   vTimer_t vTimer;
   kTimer_t kTimer;
   kTimer_t watchdogTimer;
   vClock_t vClock;
   // this field is accessible
```

Adopted Formalism

- A range of formalisms were surveyed in order to assess them for suitability
- Chosen: Isabelle/HOL
  - Used by NICTA for sel4 kernel verification
  - Well-established, industrial strength, very trusted proof-kernel

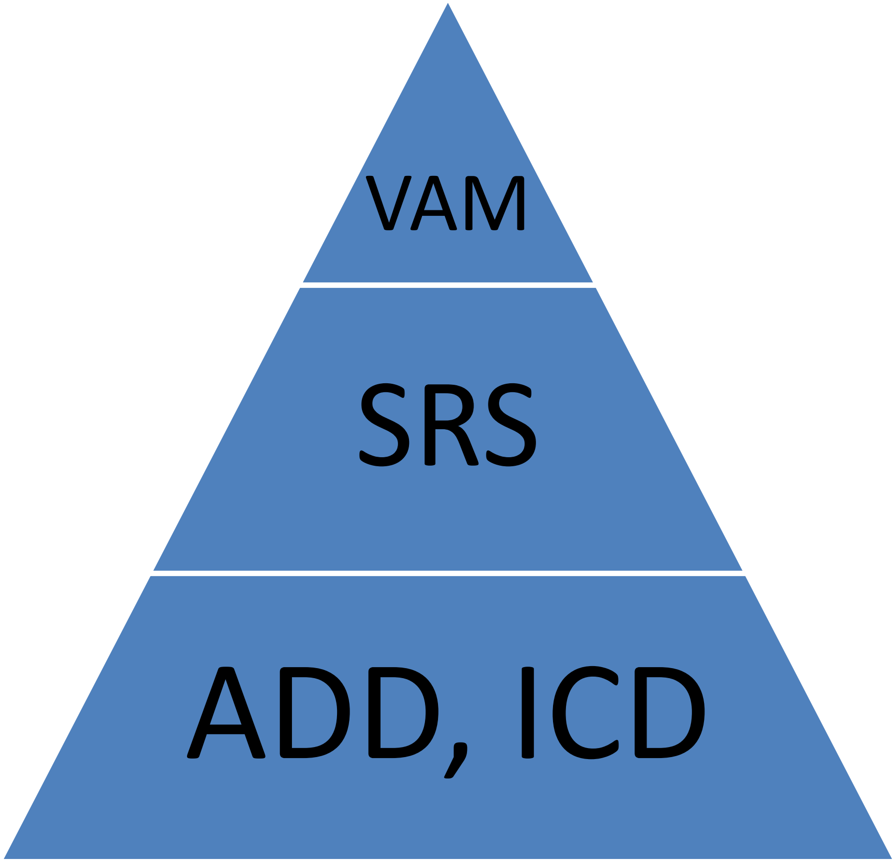
Tool-Chain:

Isabelle/HOL 2013
AutoCorres 0.9b1 (incl. C-Parser)
XtratuM v3.3-1.3 (modified)

Formal Model Hierarchy

Very Abstract Model:
high-level overview of scheduler, partitions and security policy.

Specification Model:
Provides a *high detail view* of partitions, scheduler, inter-partition communication, trap manager, security functions…



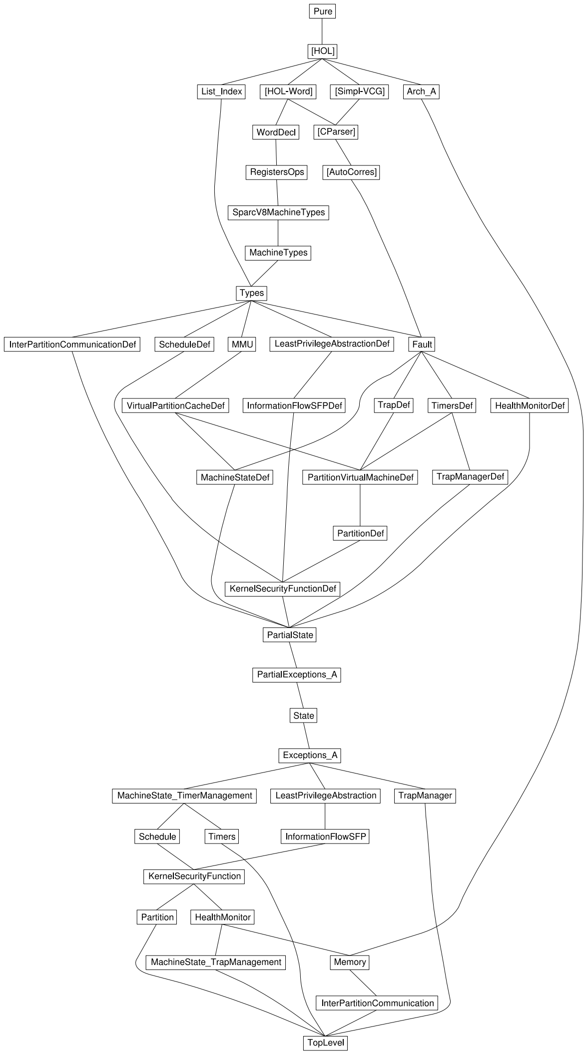
Formal Model Structure

pre-existing infrastructure, and
data-type definition theories (4 theory files)

state definitions (19 theory files).

behavioural theories
and *top-level entry point* theory (15 theory files)

38 theories, 200 data types, 600+ definitions, 9200+ LOC



From the specification to the implementation (2)

Reference Specification

Requirement Partition Control (PM_PC)
From ARINC 653P1-3 2.3.1.2

The kernel shall control and maintain the current state of each partition by means of a partition control table storing the partition state. The partition control table maintains the following list of attributes for each partition:
Partition current state.
Partition initialization reason.

Requirement Partition Current State (PM_PCS)
From IMA-SP SEP SSS v.1.1
IMA_SEP_900, IMA_SEP_901, IMA_SEP_902, IMA_SEP_903, and IMA_SEP_904

Abstract Specification

```
header{* Partition Management
Definitions *}

theory PartitionDef
imports Main Types
PartitionVirtualMachineDef
begin

text{*
  Partitions are defined according to the
Architecture Design Document
\cite{MTOBSE-ADD}
*}

section{* Partition structure *}
```

C Code

```
#ifndef _XM_KTHREAD_H_
#define _XM_KTHREAD_H_

#include <assert.h>
#include <guest.h>
#include <ktimer.h>
#include <xmconf.h>
#include <xmef.h>
#include <objdir.h>
#include <arch/kthread.h>
#include <arch/atomic.h>
#include <arch/irqs.h>
#include <arch/xm_def.h>
#ifdef CONFIG_OBJ_STATUS_ACC
#include <objects/status.h>
#endiif
```

We now have an intermediate abstract model

Abstract Consistency/Validation

Reference Specification

Requirement Partition Control (PM_PC)
From ARINC 653P1-3 2.3.1.2

The kernel shall control and maintain the current state of each partition by means of a partition control table storing the partition state. The partition control table maintains the following list of attributes for each partition:
Partition current state.
Partition initialization reason.

Requirement Partition Current State (PM_PCS)
From IMA-SP SEP SSS v.1.1
IMA_SEP_900, IMA_SEP_901, IMA_SEP_902, IMA_SEP_903, and IMA_SEP_904

?

Abstract Specification

```
header{* Partition Management
Definitions *}

theory PartitionDef
imports Main Types
PartitionVirtualMachineDef
begin

text{*
 Partitions are defined according to the
Architecture Design Document
\cite{MTOBSE-ADD}
*}

section{* Partition structure *}
```

?

C Code

```
#ifndef _XM_KTHREAD_H_
#define _XM_KTHREAD_H_

#include <assert.h>
#include <guest.h>
#include <ktimer.h>
#include <xmconf.h>
#include <xmef.h>
#include <objdir.h>
#include <arch/kthread.h>
#include <arch/atomic.h>
#include <arch/irqs.h>
#include <arch/xm_def.h>
#ifdef CONFIG_OBJ_STATUS_ACC
#include <objects/status.h>
#endiif
```

Model Verification to check:

- Data Consistency (Fully covered).
- System Calls (To be carried out).
- System Invariants (Explored).

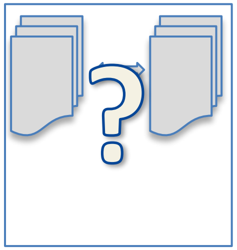
Domain expert validation to check requirements and model.

Formal Model: Validation

Req C_CC (Channel Communication)

"The kernel shall perform the inter-partition communication through channels that define logical links between one source port and one or more destination ports. The channel defines the communication transfer mode. A channel can only communicate on ports implementing the transfer mode defined in the channel."

```
datatype channel
  = Channel_Sampling channel_id port_id "port_id list" msg
  | Channel_Queuing  channel_id port_id port_id "msg_queue option"
```

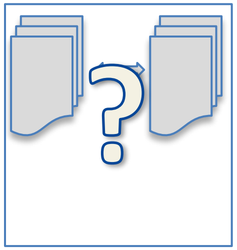


Formal Model: Validation

Req C_CC (Channel Communication)

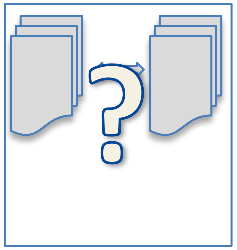
“The kernel shall perform the inter-partition communication through channels that define logical links between one source port and one or more destination ports. The channel defines the communication transfer mode. A channel can only communicate on ports implementing the transfer mode defined in the channel.”

```
definition channel_correct::"channel ⇒ communication ⇒  bool"
where
"channel_correct ch c ≡
         let ports = get_ports ch c in
         case ch of
         (Channel_Sampling _ p1 p2 _) ⇒
             port_is_sampling (fst ports)
             ∧ (∀p ∈ set (snd ports).port_is_sampling p)
         | (Channel_Queuing _ p1 p2 _) ⇒
             port_is_queuing (fst ports)
             ∧ (∀p ∈ set (snd ports).port_is_queuing p)
"
```



Formal Model: Proofs

```
lemma scheduler_ptw_limit:
"⦃λs.
    ((ksf_current_ptw (schedule (ksf s))) <
(length (get_major_time_frame (schedule (ksf s)))))
        ⦄
 scheduler
⦃ λ _ s.(ksf_current_ptw (schedule (ksf s))) < (length (get_major_time_frame
(schedule (ksf s))))⦄"
 unfolding scheduler_def
 apply (simp  add: nextPTW'_def get_major_time_frame_def )
 apply (simp add:  getNextTimeFrameWindow_def getCurrentPartitionId_def)
 apply (simp_all add: Let_def)
 apply (simp_all add: liftS_def dummyp_def ps_2_s_def s_2_ps_def)
 apply (monad_eq)
 apply (simp_all add:partial_state.defs state.defs)
 apply (simp_all add:valid_def)
 apply (monad_eq)
 apply auto
 done
```



Code Correctness

Reference Specification

Requirement Partition Control (PM_PC)
From ARINC 653P1-3 2.3.1.2

The kernel shall control and maintain the current state of each partition by means of a partition control table storing the partition state. The partition control table maintains the following list of attributes for each partition:
Partition current state.
Partition initialization reason.

Requirement Partition Current State (PM_PCS)
From IMA-SP SEP SSS v.1.1
IMA_SEP_900, IMA_SEP_901, IMA_SEP_902, IMA_SEP_903, and IMA_SEP_904

?

Abstract Specification

```
header{* Partition Management
Definitions *}

theory PartitionDef
imports Main Types
PartitionVirtualMachineDef
begin

text{*
  Partitions are defined according to the
Architecture Design Document
\cite{MTOBSE-ADD}
*}

section{* Partition structure *}
```

?

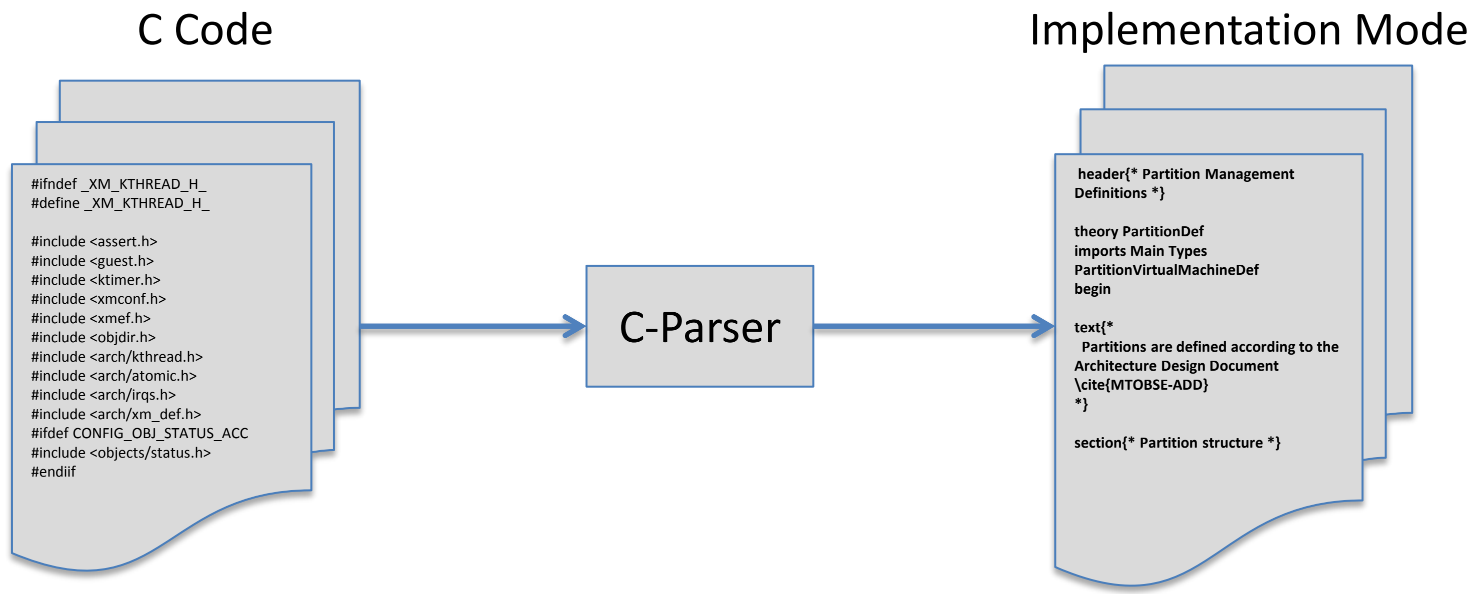
C Code

```
#ifndef _XM_KTHREAD_H_
#define _XM_KTHREAD_H_

#include <assert.h>
#include <guest.h>
#include <ktimer.h>
#include <xmconf.h>
#include <xmef.h>
#include <objdir.h>
#include <arch/kthread.h>
#include <arch/atomic.h>
#include <arch/irqs.h>
#include <arch/xm_def.h>
#ifdef CONFIG_OBJ_STATUS_ACC
#include <objects/status.h>
#endif
```

Implementation Model.
Proof of Refinement: Create a refinement model.
Proof of implementation invariants.

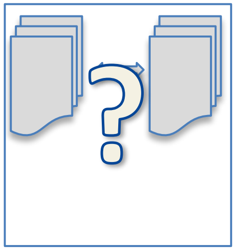
Implementation Model



Input: C-99 subset

- No side effects in expressions.
- No reference to local variables.
- No unions and bitfields.
- No "goto" C statements.

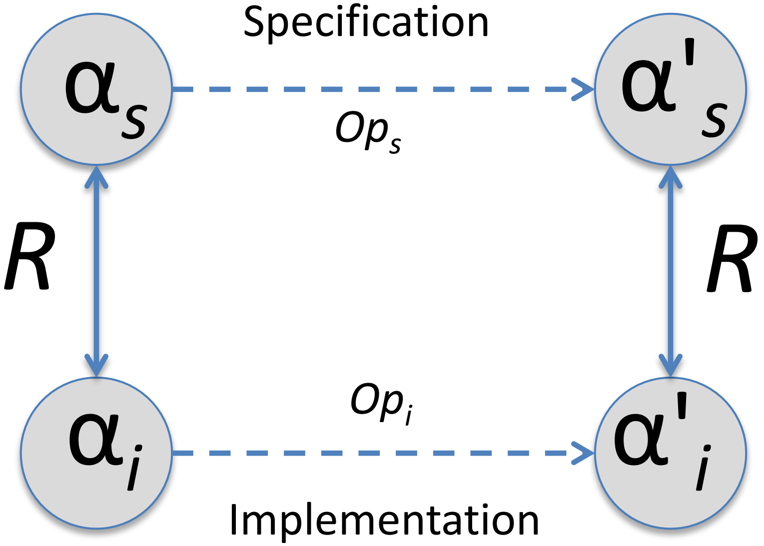
Output: Isabelle definitions and theorems.

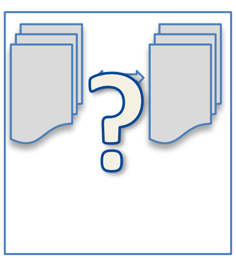


Used XtratuM code as a test case – we had to modify it to fit the subset

Model Refinement

M_i refines M_s





Refinement uses a relation *R* that links specification states to corresponding implementation states.

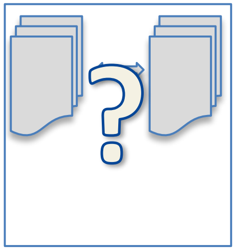
We say that M_i refines M_s when for all corresponding specification and implementation operations (Op_s,Op_i), both transform initial states related by R ($\alpha_s \, R \, \alpha_i$) to final states that are themselves so related ($\alpha'_s \, R \, \alpha'_i$).

When M_i refines M_s, the properties that hold in M_s also hold in M_i

Refinement (SW02)

$$
\begin{aligned}
\mathcal{R}^{IPC} :: \ & communication\_state \Rightarrow xmcCommPort[\,] \Rightarrow xmcCommChannel[\,] \\
& \Rightarrow xmcPartitionTab[\,] \Rightarrow union\ channel \Rightarrow struct\ port \\
& \Rightarrow struct\ xmc \Rightarrow string[\,] \Rightarrow Boolean
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{R}^{IPC}\ & a\_cs\ i\_p\_conf\ i\_c\_conf \\
& i\_part\_tab\ i\_p\_tab\ i\_c\_tab\ i\_xmc \\
& xmcStringTab \equiv \\
& \quad (\mathsf{nochannels}\ xmc) = |\ \{ch\_id.(\mathsf{channels}\ a\_cs)\ ch\_id \neq None\}\ | \\
& \quad \wedge \forall i\_ch. 0 \leq i\_ch < (\mathsf{nochannels}\ xmc), \\
& \qquad \exists a\_ch\ \exists ch\_id.(\mathsf{channels}\ a\_cs)\ ch\_id = Some\ a\_ch \\
& \qquad \wedge\ a\_ch\ \mathcal{R}^{Channel}\ (i\_c\_conf!\, i\_ch\ i\_c\_tab!\, i\_ch) \\
& \quad \wedge \forall i\_p. 0 \leq i\_p < (\mathsf{noports}\ xmc) \wedge (\mathsf{channel\_id}\ i\_p\_conf!\, i\_p) = i\_ch, \\
& \qquad \exists a\_p\ \exists p\_id.(\mathsf{ports}\ a\_cs)\ p\_id = Some\ a\_p \\
& \qquad \wedge\ (\mathsf{port\_id}\ a\_p) \in (\mathsf{ports}\ a\_ch) \\
& \qquad \wedge\ a\_p\ \mathcal{R}^{Port}\ (i\_p\_conf!\, i\_p \\
& \qquad\qquad\qquad i\_p\_tab!\, i\_p \\
& \qquad\qquad\qquad i\_part\_tab!\, (\mathsf{partition\_id}\ (i\_p\_tab!\, i\_p)) \\
& \qquad\qquad\qquad xmcStringTab)
\end{aligned}
$$



Vendor Verification Process

1. Bring vendor specification and code into alignment with Reference Specification and ESTEC/formal coding standards.
2. Build formal model of vendor code
3. Formalise invariants for the vendor code.
4. Formalise refinement between code and specification
5. Prove refinement respects invariants.
6. Prove all API calls preserve invariants.
7. Prove all API calls satisfy corresponding abstract API specification.

Invariants are key !

Artifact Families

Common Criteria defines various families of Development artifacts.

Below we list these with approximate matching MTOBSE deliverables:

Family	MTOBSE Deliverable
SPM: Security Policy Model	Very Abstract Model
FSP: Functional Specification	Reference (SRS) & Abstract Specification
HLD: High-Level Design	Reference (ADD,ICD) & Abstract Specification
LLD: Low-Level Design	-
IMP: Implementation Representation	Modified XtratuM code and C-Parser generated model
RCR: Representation Correspondance	Added to C-Parser generated model.

Levels of Formality

Level	Informal	Semi-formal	Formal	Proof
EAL5	_	FSP HLD RCR	SPM	
EAL6	_	FSP HLD LLD RCR	SPM	
EAL7	_	LLD	SPM FSP HLD RCR	RCR
SKPP	FSP	LLD	SPM FSP HLD RCR	RCR

EAL: Common Criteria Evaluation Assurance Level
SKPP: Separation Kernel Protection Profile for High Robustness

"Informal": A natural language description, written carefully and unambiguously
"Semi-Formal": Using a language with a formal syntax and well-defined semantics (e.g., UML).
"Formal": Using a language with a formal syntax and a mathematically defined semantics.

Feasibility, Cost (CC,SKPP)

Level	Person-Months	Notes
EAL5	7+	Mainly refining and validating SW02
EAL6	25+	Nature of LLD unknown to us
EAL7	43+	
SKPP	43+	Both D03 *and* SW02 are needed

All estimates are provisional, and depend on *continuity of expertise* and *control of all tools and models*.

None of the above require code (IMP) to be formally verified, but this has been done by others, e.g, NICTA and sel4.

Feasibility, Cost (Code Verification)

- NICTA's costs
  - Abs. Spec. - 4 person-months
  - Haskell Prototype - 2 person-years
  - Executable Spec - 3 person-months
  - C implementation - 2 person-months
  - Proof R&D - 9 person-years
  - Proof of seL4 - 11 person-years

They could leverage off accumulated skill-set and experience and redo this in 8-py (their estimate)

We see 8 person-years as a likely lower bound for TSP code verification

Achievements

- We have:
  - developed a Reference Specification
  - formalised this specification (Abstract Specification)
  - explored feasibility/costs of formal verification.
- Conclusion:
  - EAL5+ formal verification is feasible
  - Code verification is possible, but not cheap.

Thank You!

Questions?

Formalisms Assessed

- Isabelle/HOL
- Coq,
- ACL2
- PVS
- Z/Eves
- B-Method
- VCC
- TLC

Very Abstract Model

Very Abstract Model: Provides a very abstract interpretation of the scheduler, partitions and the security policy.

type-synonym $PId = nat\ set$
type-synonym $ActPol = PId \rightharpoonup Action\ set$
type-synonym $RunHist = (PId \times Action)\ list$
type-synonym $Run = Action\ list$
type-synonym $Apps = PId \rightharpoonup Run$
type-synonym $Sched = PId\ list$
type-synonym $Sys = (ActPol \times Sched \times PId \times Apps)$

definition $appActs :: Apps \Rightarrow Action\ set$
definition $invSys :: Sys \Rightarrow bool$

function $execute :: ActPol \Rightarrow (Sched \times Apps) \Rightarrow RunHist$
and $perform :: (ActPol \times Sched) \Rightarrow (Apps \times PId) \Rightarrow Run \Rightarrow RunHist$

definition $run :: Sys \Rightarrow RunHist$
definition $RunHistofP :: PId \Rightarrow RunHist \Rightarrow Run$
definition $issublist :: 'a\ list \Rightarrow 'a\ list \Rightarrow bool$

lemma $policy$: $invSys(\alpha,\sigma,pf,\beta) \Longrightarrow hpConsistent\ \alpha\ (run\ (\alpha,\sigma,pf,\beta))$