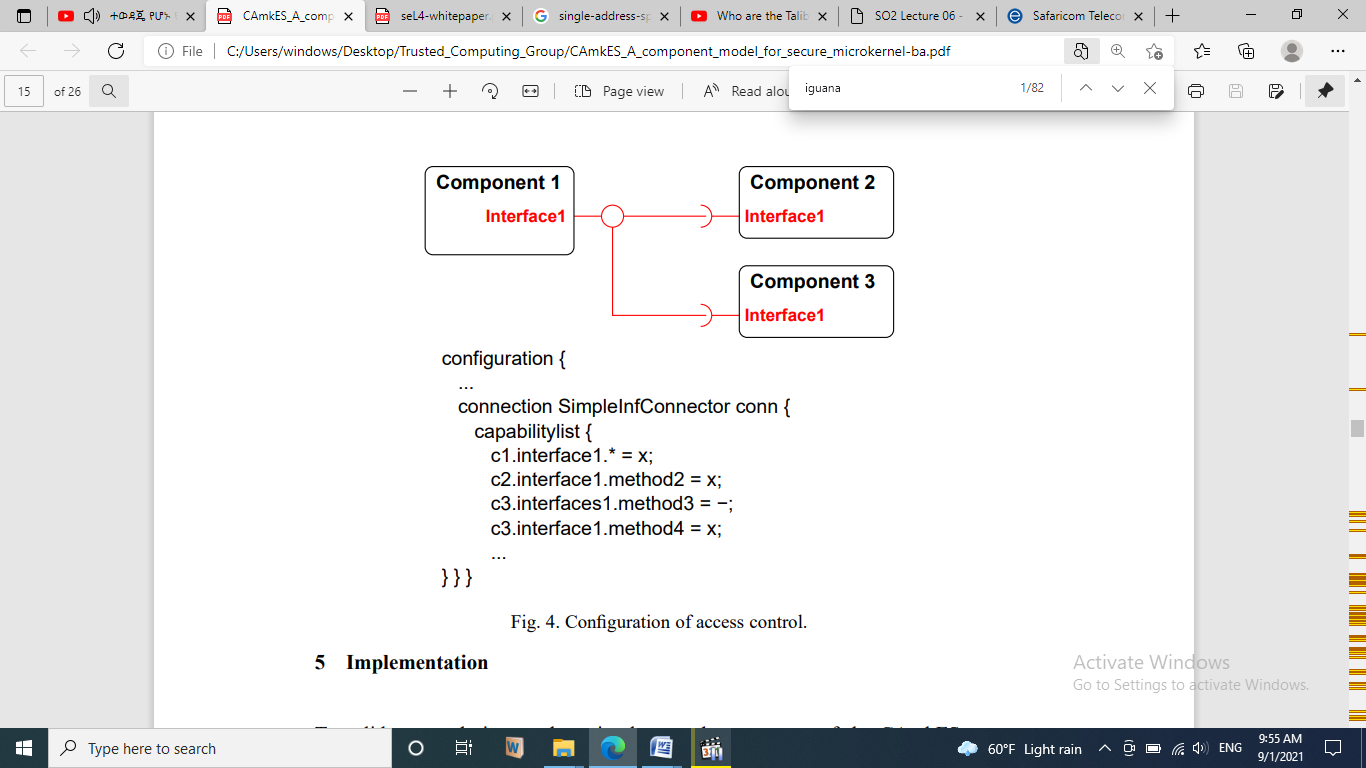
Iguana (The Supervisory OS)

The basic security model provided by CAmkES is based on Iguana’s capability model and involves controlling and restricting access to components. In particular we use configuration specifications to specify a capability list in the configuration of a connection.

Figure 4 shows a scenario where Component 1 provides interfaces to Component 2 and Component 3. In this example, c2.interface1.method2=x means that Component 2 can access method2 of interface1 provided by Component 1. Similarly, c3.intface1.method3=- means Component 3 cannot access method3 defined in interface1 provided by Component 1. We can see from the configuration of the capability list that Component 2 is given permission to invoke method2 defined in interface1, while Component 3 does not have the right to invoke method3, but can invoke method4. These access restrictions are enforced by Iguana at runtime.

Note that by relying on mechanisms already provided by Iguana, we can support this model without adding significant overhead to the CAmkES runtime.

Furthermore, since the access control is part of the functionality of a connector, it is possible to use different access control mechanisms by using different connectors



In particular, since L4 also avoids implementing policy, it does not provide any specific model of operating system services such as process management, memory and address space management, access control, etc. This task is left up to a supervisory OS running in user-mode on top of the microkernel. In our case this OS is Iguana.

Iguana is specifically designed for use in embedded systems. It has low memory and cache footprints and provides basic services such as memory management, protection management, a remote procedure call (RPC) based IPC mechanism, lowoverhead data-sharing and a basic device driver framework.

Iguana provides a single non-overlapping address space that is shared by all threads. In Iguana the concerns of memory protection and memory translation (i.e., providing address spaces) are separated. This means that despite all threads sharing the same address space, Iguana also provides memory protection.

The separation of memory protection and translation also means that Iguana-based systems can be readily deployed on processors without virtual memory. Moreover, this allows increased performance to be gained on processors (such as ARM7 and ARM9) with virtually-addressed caches where an overlapping address space layout would require a cache flush on every context switch.

Iguana provides a client-server model of interaction. Applications and operating system services run as Iguana servers and interact with each other using IPC.

An Iguana server consists of a thread with an associated memory section running in a protection domain. Threads are Iguana’s basic units of execution and scheduling, and memory sections are the basic units of virtual memory allocation and protection.

Protection domains provide memory protection between threads executing different programs (or servers). A protection domain roughly corresponds to the concept of a task or process in other systems, except that a protection domain does not define a separate virtual address space.

Threads in the same protection domain have full access to each others memory, while threads in different protection domains are protected from each other and can access each others memory only if permitted to by the access control system.

This is implemented using capabilities, which are security tokens that define access rights to memory sections and threads. Thus, in order to access a memory section in another protection domain a thread must hold an appropriate read or write capability for that memory section.

Each Iguana server implements a server-specific interface that consists of a set of methods that can be invoked on that server.

Iguana provides a remote-procedure call style of IPC.

A client invokes a server’s method by calling a local stub function. The stub marshals parameters and sends a message to the server using underlying L4 IPC mechanisms. At the server side, the message parameters are un-marshaled by a similar stub and the appropriate function is invoked.

Before invoking another server’s methods, a session must be established between the client and server. Besides setting up a communication channel, establishing a session also involves ensuring that the communicating parties hold the right capabilities.

In order to invoke a method on a server in another protection domain, the invoker must hold an appropriate execute capability for that server.

5.2 **Mapping CAmkES to Iguana**

In order to run CAmkES components on top of Iguana we provide a mapping of CAmkES concepts onto Iguana concepts.

CAmkES components are generally placed in separate Iguana protection domains and are implemented as separate Iguana servers. This provides proper encapsulation and prevents other components (or processes) from purposefully or inadvertently accessing a component’s internals. Furthermore, it allows the architecture to restrict access to a component’s interfaces to authorized parties only.

Note that the underlying OS makes use of hardware-based memory protection to enforce this.

CAmkES RPC interfaces map indirectly to Iguana interfaces. Unlike CAmkES interfaces, Iguana interfaces act as units of protection rather than encapsulation. In order to provide method-level access control this means that, when mapping to Iguana, the individual methods of a CAmkES RPC interface are translated to separate Iguana interfaces. We call these the Iguana equivalent interfaces.

In our prototype implementation, the components are active and contain dispatch threads that allow them to service Iguana RPC requests.

CAmkES dataports map to shared Iguana memory sections so the sharing of memory sections in Iguana is managed by the memory management (or protection) unit and does not require any copying of data. CAmkes events map to Iguana asynchronous notifications, however, since the Iguana implementation of these is currently in a state of flux, they have not been included in the prototype.

Connections are mapped according to the interfaces that they connect. RPC connections naturally result in Iguana IPC communication. This is managed by stubs generated from Iguana IDL descriptions of the connected interfaces’ Iguana equivalents. Dataport connections are implemented as shared Iguana memory sections. Dataport initialization code takes care of setting up the memory sections and mapping these onto appropriate local variables in the relevant components.

Compound components do not map directly onto any Iguana entities. Since a compound component contains other components, but does not implement any functionality itself, it is not necessary to have a separate entity representing it. Instead, any access to a compound component’s interface is routed directly to the component actually implementing that interface.

Loading and initialising a CAmkES-based system proceeds roughly as follows:

* A boot image containing L4, Iguana, and CAmkES components is loaded into the system’s memory.
* L4 starts and loads the Iguana user-mode server.
* Once loaded, the Iguana server proceeds to load and initialize its services.
* After basic services such as chipset drivers, naming, etc. have been loaded, a CAmkES loader routine is run. The loader routine is responsible for loading all components, initializing them and establishing connections between them.
* Connection establishment involves the creation of Iguana sessions, allocation of shared memory sections and the distribution of capabilities according to the component configuration specifications.
* Finally, once all components and connections have been initialized, component dispatch and control threads are started.

**Evaluation**

..................

Abbreviation

IDL: - interface definition language

Note:

Besides resource restrictions, the safety and security properties of embedded system are of utmost importance. These are addressed in our model through a tight integration with an underlying secure microkernel-based operating system (L4/Iguana).

L4/Iguana, which has been developed specifically for safe and secure embedded systems, provides protection mechanisms such as capability-based access control to encapsulate complex software into protected components.

Given our static model, it would also be possible to analyze a composition at build time to ensure that no access restrictions are violated. In such a situation, the runtime access controls would not be required, which would save much runtime overhead. This approach to safety and security has not yet been followed up, but is something that we wish to look into in the future.

Dictionary

Low overhead: - The costs of running the business are relatively low.

State of flux: - A state of uncertainty about what should be done (usually following some important event) preceding the establishment of a new direction of action