Realtime Image Analysis with CORBA in an Industrial Production Environment

Lothar Werzinger

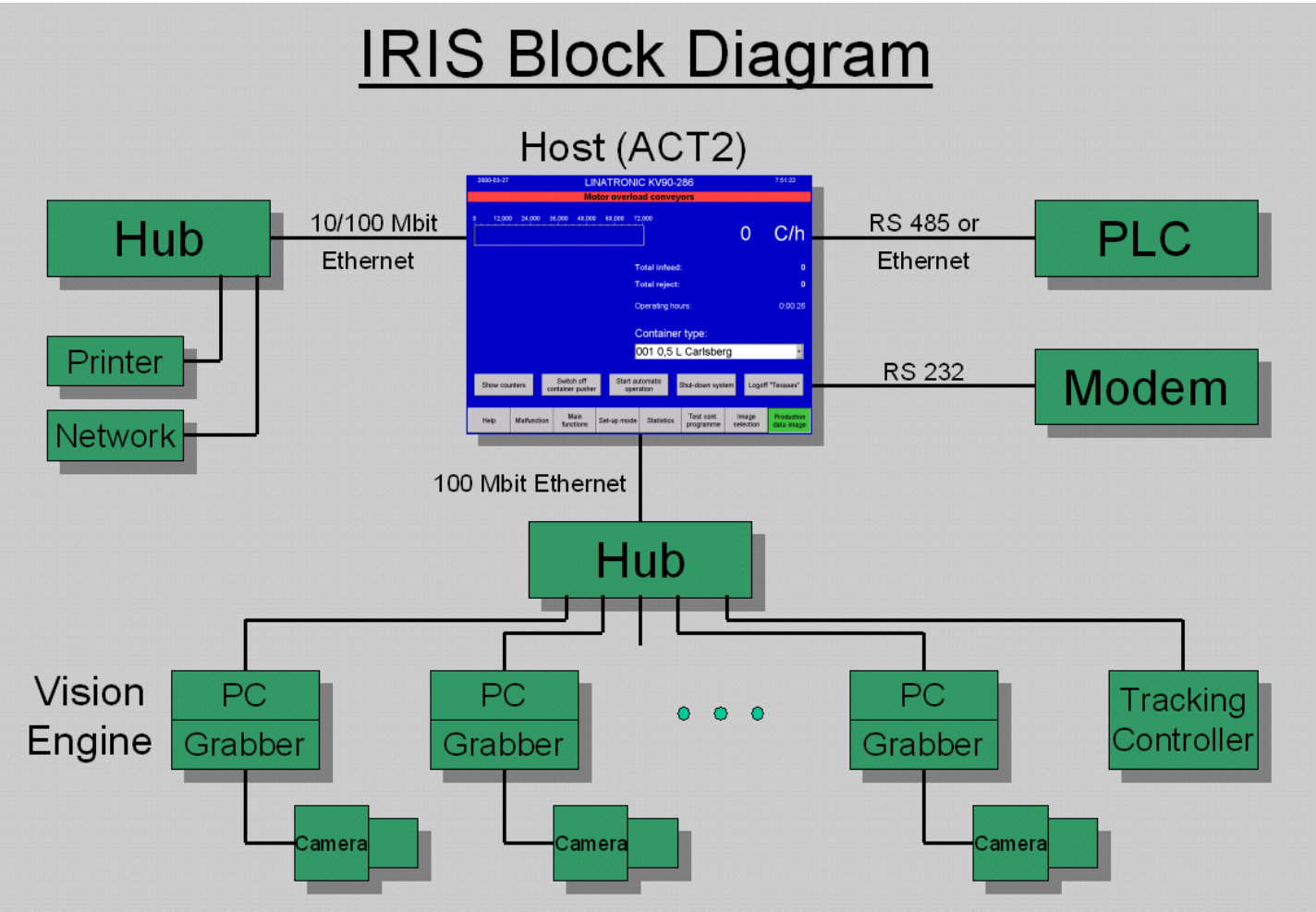
Dipl.-Ing. Univ.
werzinger.lothar@krones.de
phone: ++49 (+94 01) 70-34 99
fax: ++49 (+94 01) 70-36 79

R&D Electronic Development
Software Technologies
Krones AG
www.krones.de

You can get these slides via www.krones.de/download/realtime_workshop_2001[.pdf/_2.pdf/_4.pdf]

Inspection machine Linatronic M2

Improved Resolution Inspection System Architecture



Design Requirements

- Modular design
  - Component based
  - Extensible
  - Run time configurable
- Distributed system
  - Windows NT HMI terminal with Java GUI
  - Linux based embedded inspection units ($\geq$ 10)
  - Inspection Software runs on HMI terminal, too
- Realtime constraints
  - Analyze $\geq$ 40 Images per second

General Information

- ORBs used
  - TAO (C++ side)
  - JacORB (Java side)
- Components
  - Framework based model
  - Components based on a basic IDL interface
  - Individual analysis modules ($\geq$ 25)
- Result propagation uses TAO RTEC (later Notfication Service)
- Deployed more than 50 machines
- More than 300000 lines of code

Patterns used

Uncomplete list of pattern used in our software

- Factory
- Dynamic Configuration
- Strategy
- Delegation
- Wrapper Facades
- . . .
- Scoped Locking
- Double-Checked Locking
- Active Object
- Servant Locator
- Reactor

Design of Interfaces

- Design for realtime $\neq$ design for non realtime
- Design of interfaces can affect performance significantely
- Take language mappings into account
- If appropriate modify existing designs
  - based on new information you have
  - even, if the new design looks more complicated
  - by analyzing if there is more than one way to do something

Sample IDL (first try)

Simple IDL of image

```
interface Image
{
  typedef sequence < octet > ImageData;

  ImageData    getData ( );
};
```

Implementation

Sample implementation of image

```
    const unsigned char *           pc_data;
    long                            dataSize;
    ACE_Mem_Map                     Map;
    ImageData_var                   mv_data;

    Map.map( "filename" );
    pc_data  = Map.addr();
    dataSize = Map.size();

    mv_data = new ImageData(
      dataSize, dataSize,
      const_cast<CORBA::Octet *>(pc_data), 0
    );

  ImageData * Image::getData ( ) // shallow copy {
    ImageData_var v_result = new ImageData(
      mv_data->length(), mv_data->length(),
      &(mv_data[0]), 0
    );
  }
```

Sample IDL (second try)

More advanced IDL of image

```
interface Image
{
  enum ImageKind
  {
    eGray8,  // 8 bit grayscale image.
    eGray16  // 16 bit grayscale image.
  };

  union ImageData switch ( ImageKind )
  {
    case eGray8:
      sequence < octet >            m_data8;
    case eGray16:
      sequence < unsigned short >   m_data16;
  };

  ImageData    getData ( );
};
```

C++ mapping for Unions

Excerpt from the C++ mapping for ImageData Union

```
class ImageData {
  ...

  // set
  void m_data8 ( const _tao_seq_Octet & );

  // get (read only)
  const _tao_seq_Octet & m_data8 ( void ) const;

  // get (read/write)
  _tao_seq_Octet & m_data8 ( void );
};

class ImageData_var {
  ...
    // copy constructor makes deep copies
  ImageData_var (const ImageData_var &); // copy constructor
};
```

C++ mapping for Unions cont` d

Excerpt from the C++ implementation for ImageData Union

```
ImageData_var::ImageData_var (const ImageData_var &p) // copy constructor
{
  if (p.ptr_)
    // copy constructor makes deep copies per definition
    this->ptr_ = new ImageData (*p.ptr_);
  else
    this->ptr_ = 0;
}
```

Sample IDL (final)

More advanced IDL of image

```
interface Image
{
  enum ImageKind
  {
    eGray8,  // 8 bit grayscale image.
    eGray16  // 16 bit grayscale image.
  };

  union ImageData switch ( ImageKind )
  {
    case eGray8:
      sequence < octet >            m_data8;
    case eGray16:
      sequence < unsigned short >   m_data16;
  };

  getData ( inout ImageData ar_imageData );
};
```

Implementation

Sample implementation of enhanced image

```
void Image::getData ( ImageData & ar_imageData )
{
  switch(m_kind)
  {
    case eGray8:
      ar_imageData.m_data8(ImageData::_m_data8_seq());
      ar_imageData.m_data8().replace (        // make a shallow copy
        mv_data->length(), mv_data->length(),
        (CORBA::Octet *)(&(mv_data[0])), 0
      );
      break;
    case eGray16:
      ar_imageData.m_data16(ImageData::_m_data16_seq());
      ar_imageData.m_data16().replace (       // make a shallow copy
        mv_data->length(), mv_data->length(),
        (CORBA::UShort *)(&(mv_data[0])), 0
      );
      break;
  }
}
```

Sequences

```
SomeItem * doSomething ( )
{
   SomeItem_var v_return = new SomeItem ();
   v_return->m_xyz = 42;
   return(v_return._retn());
}

 ...

SomeSequence  seq;

 ...

SomeItem_var v_result;
while( goOn() ) {
  seq.length(seq.length() + 1);
  v_result = doSomething();
  seq[seq.length() - 1] = *v_result;
}
```

Enhanced use of Sequences (TAO)

```
void doSomething ( SomeItem & ar_item )
{
   ar_item.m_xyz = 42;
}

 ...

SomeSequence  seq;

// preallocate someLimit number of items
seq.length(someLimit);
// set real size back to 0
seq.length(0);

 ...

while( goOn() ) {
  seq.length(seq.length() + 1);
  doSomething(seq[seq.length() - 1]);
}
```

Concluding Remarks - pro` s

- Design interfaces especially for realtime use
- Redesigning interfaces can be a huge speedup
- Considerations to be taken:
  - What implication come from the language mapping?
  - Can the design be modified to be more efficient, even if it looks more complicated?
  - Are there more than one ways to achieve the task?
- Search for optimisations in the implementation

Concluding Remarks - con` s

- Speed optimized interfaces tend to be more complex and harder to understand
- Optimisations for one language mapping may slow down another one
- Using implementations in some optimized ways can be non portable
- Does the speed up really pay for a more complicated interface

Patterns and Frameworks Literature

Books

- Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software* AW, ’94
- *Pattern Languages of Program Design* series by AW, ’95-’99.
- Siemens & Schmidt, *Pattern-Oriented Software Architecture: A System of Patterns*, Wiley, vol. ’96 & ’00, ISBN 0-471-95869-7
- Siemens & Schmidt, *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*, Wiley, 2000, ISBN 0-471-60695-2.

Special Issues in Journals

- October ’96 CACM (guest editors: D. Schmidt, R. Johnson, and M. Fayad)
- October ’97 CACM (guest editors: D. Schmidt and M. Fayad)

Magazines

- C++ Report, *e.g.*, columns by Coplien, Vlissides, Schmidt & Vinoski, and Martin

How to Obtain ACE & TAO Software

- The ADAPTIVE Communication Environment (ACE) is an OO toolkit designed according to key network programming patterns
- The ACE ORB (TAO) is a real-time, high-performance ORB. Fully compliant with OMG 2.2 / 2.3 spec.
- All source code for ACE & TAO is freely available
  - www.cs.wustl.edu/∼schmidt/ACE.html
  - www.cs.wustl.edu/∼schmidt/TAO.html