Scope of Component Models

- Component Models for configuring and assembling application level components
- Weaving of components done based on application characteristics
- Component configuration isolated from component development
- Some level of automation
- Serves reusability of components
Need for Finer-level Configuration

- Component configurations
  - Inter-Component configuration
  - Intra-component configuration
    • Component Implementation infrastructure

- Mapping of
  - High level component configurations to
  - Low level infrastructure elements

- Configuration of low-level infrastructure elements involve
  - Selection of various strategies based on high level configurations
  - Validating appropriate combinations of configurations
Static vs. Dynamic Configuration

- Static configuration chosen at system startup
  - Compile-time decisions to improve performance
  - Compile-time checks based on system properties known \textit{a priori}
  - Inference of strategies based on high level configurations

- Reconfiguration during system mode changes
  - Expensive, but more flexible and dynamic
  - E.g. Factory based configurations

- Hybrid strategies
  - As a balance between the two
Generative Approach as a Solution

- Generative Programming
  - Focus on families of software systems
  - Concrete system generated based on configuration knowledge

- Extend Generative programming approach to low-level infrastructure configuration
  - Generate low level configurations from high level configurations automatically

- Configuration generators
  - For mapping high level to low level configurations
  - For assembling and weaving the right infrastructure elements
C++ Template Meta-Programming

• Mechanism to embed generators in C++
  – Completely within the purview of C++ language

• Metainformation represented using
  – Member traits, Traits classes, Traits templates

• Compile-time control-flow constructs
  – Template metafunctions E.g. IF, THEN, ELSE, CASE
  – Conditional compilation based on evaluation of type-expressions

• Issues
  – Advanced usage of C++ templates
  – Compiler support an issue
Example – RT Dispatching

Threads are all from dispatcher
- Timers trigger suppliers
- Suppliers push to EC
- Events land in queues
- Worker threads pull from queues and push to consumers
RT Dispatching Configuration

• QoS attributes based on scheduling policy
• Bundle together all QoS attributes in one descriptor
• Can we generate the appropriate QoS descriptor?
  – Use a configurator to generate the attributes
  – Scheduling policy as input to generator

Scheduling policy

QoS Descriptor Generator

QoS Descriptor
enum Disp_Rule_t
    { RMS, EDF, MLF, MUF, other }

template<Disp_Rule_t>
struct QoSDesc
{
};
//template specializations

template<>
struct QoSDesc<RMS>
{
    long period;
    //fields specific to RMS
};
template<>
struct QoSDesc<EDF>
{
    long deadline;
    //fields specific to EDF
};
template<>
struct QoSDesc<MLF>
{
    //fields specific to MLF
};

template <Disp_Rule_t disp_rule>
struct QoSDescriptorGenerator
{
    typedef typename
        CASE<EDF,QoSDesc<EDF>,
        CASE<RMS,QoSDesc<RMS>,
        CASE<MLF,QoSDesc<MLF> > >
        disp_rule_case_list;

    typedef typename
        SWITCH<disp_rule,
        disp_rule_case_list>::RET
        QoSDescriptor_;

    typedef QoSDescriptor_ RET;
};

typedef QoSDescriptorGenerator<EDF>::RET
    QoSDescriptor;
Other Use-Cases

- ORB Infrastructure configuration
  - Strategies based on inference from system properties
- E.g. Strategy used to wait for replies
  - Wait on connection
    - No interleaved processing of incoming requests
    - No blocking factor
  - Wait on reactive mechanism
    - Interleaved processing of incoming requests
    - Blocking factor needs to be considered
- Configurator to choose strategy based on system characteristics

Nested Upcall scenario
Related Work

• RMA using template meta programming
  – RMA schedulability analysis at compile-time within the C++ type system
  – Compile error when utilization bound exceeds the RM utilization bound

  “Rate-Monotonic Analysis in the C++ Typesystem”, Deters, Gill and Cytron, presented at RTAS 2003 Workshop on Model-Driven Embedded Systems

• Task-Scheduler Logic
  – Work by John Regehr and Alastair Reid
  – Reasoning about Concurrency in Component-Based Systems Software
  – First order logic for representing system knowledge
Conclusions

• Component models deal with higher level application components
• Need to map high level specifications to configuration of finer-grained components
• Extend generative programming to configuration of fine-grained infrastructure
• Algebraic specification of system behavior
• Inference of infrastructure configuration strategies by reasoning of system behavior