Modeling Guidelines and Refinement Rules

Gajski, et al., *SpecC* textbook, Chapter 4, “The SpecC Methodology”

Model Guidelines

- Specification
- Architecture
- Communication
- Implementation

Refinement Rules

- Behavior partitioning
- Scheduling
- Variable partitioning
- Channel partitioning
- Protocol insertion
- Transducer synthesis
- Inlining

Summary Listing of Guidelines and Rules

Table 4.1. Specification model guidelines.
- Separate communication and computation
- Expose parallelism
- Use hierarchy to group related functionality
- Choose proper granularity
- Identify system states

Table 4.2. Refinement rules for behavior partitioning.
- Introduce additional level of hierarchy
- Bind behaviors to components
- Group behaviors
- Estimate behavior metrics
- Add synchronization
- Move communication

Table 4.3. Refinement rules for scheduling.
- Serialize behavior hierarchy
- Optimize control

Table 4.4. Refinement rules for variable partitioning.
- Move variables into components
- Add communication channels
- Update variable accesses
- Optimize communication and synchronization

Table 4.5. Refinement rules for channel partitioning.
- Add level of hierarchy
- Bind channels to busses
- Group communication
- Estimate channel metrics
- Update channel accesses

Table 4.6. Architecture model guidelines.
- Create high-level structure
- Sequentialize component functionality
- Specify global communication
- Annotate estimated metrics

Table 4.7. Refinement rules for protocol insertion.
- Insert protocol code
- Generate application layer
- Replace bus channels

Table 4.8. Refinement rules for transducer synthesis.
- Insert transducers
- Encapsulate with wrappers
- Synthesize transducer code

Table 4.9. Refinement rules for inlining.
- Inline communication methods
- Optimize

Table 4.10. Communication model guidelines.
- Implement bus functionality
- Annotate bus timing

Table 4.11. Implementation model guidelines.
- Implement system busses
- Implement system components

Table 4.1. Specification model guidelines.

Separate communication and computation

Algorithmic functionality has to be detached from communication functionality. In addition, inputs and outputs of a computation have to be explicitly specified to show data dependencies.

Expose parallelism

Allow independent behaviors to run concurrently instead of artificially serializing behaviors in expectancy of a serial implementation. In essence, all parallelism should be made available to the exploration tools in order to increase room for optimizations.

Use hierarchy to group related functionality

Introduce one hierarchical level for each functional group and eliminate localized effects at higher levels. For example, local communication and local data dependencies are grouped and hidden by the hierarchical structure.

Choose proper granularity

The size of leaf behaviors has to be chosen such that optimization possibilities and design complexity are balanced when searching the design space. Basically, the leaf behaviors, which build the smallest indivisible units for exploration, should reflect the division into basic algorithmic blocks.

Identify system states

Use state transitions to explicitly model the steps of the computation in terms of basic algorithms or abstracted, hierarchical blocks.

Table 4.2. Refinement rules for behavior partitioning.

Introduce additional level of hierarchy

At the top-level of the behavior hierarchy, insert behaviors which represent the components of the system architecture.

Bind behaviors to components

Annotate the component behaviors with the name of the component type out of the component library. Since the inserted behaviors simply group behaviors for each PE, this establishes the correlation of PE

behaviors with allocated components in the system architecture.

Group behaviors

Group the behaviors of the specification under the component behavior to which they have been mapped, preserving the structural and behavioral hierarchy of the specification in the parts mapped to each component.

Estimate behavior metrics

Annotate the behaviors with the estimated values of chosen metrics for the components executing the behaviors. For example, in leaf behaviors appropriate waitfor ( ) statements are added to establish correct execution times during simulation.

Add synchronization

For original behavior transitions that now cross component boundaries, introduce additional control behaviors in each component and corresponding global synchronization channels in order to preserve execution semantics.

Move communication

Move variables and channels in the original specification that are used for communication between behaviors mapped to different components to the top level and declare them as global system variables/channels. Add corresponding ports and connections in the structural hierarchy from the top down to the behaviors accessing the variables and channels.

Table 4.3. Refinement rules for scheduling.

Serialize behavior hierarchy

Inside the PE behaviors, remove all concurrent (parallel, pipe) behaviors and transform the behavior hierarchy according to the selected schedule into a purely sequential execution. Possibly flatten parts of the hierarchy or move behaviors across the hierarchy.

Optimize control

Optimize the scheduled hierarchy by removing unnecessary control and synchronization behaviors and channels.

Table 4.4. Refinement rules for variable partitioning.

Move variables into components

According to the selected partition, move variables into the (memory or processing) components to which they have been assigned.

Add communication channels

For each variable add a global communication channel. In case a variable has been mapped to a dedicated server component, add a channel for communication with the memory server and connect all components accessing the variable to that channel. In case a local copy of a variable is kept in each component, add a message-passing channel for exchange of updated values.

Update variable accesses

For PEs with no local copy of a variable, replace all variable accesses with read ( ) and write () calls to the corresponding channel. Otherwise, replace with accesses to local copy and add code at each synchronization point (wait) to send or receive updated variable values over the corresponding message-passing channel, in case the local copy was modified before or will be needed after synchronization, respectively. In both cases, update ports and connectivity accordingly.

Optimize communication and synchronization

Optimize variable communication by merging it with any existing communication and/or synchronization. For optimization, remove communication channels and corresponding code if variable is accessed only inside the assigned component.

-

Table 4.5. Refinement rules for channel partitioning.

Add level of hierarchy

Introduce an additional level of hierarchy. At the top-level of the channel hierarchy, insert channels which represent the busses in the system architecture.

Bind channels to busses

Annotate the busses with the name of the bus type and bus protocol out of the IP library. Since the inserted busses simply group communication handled over each bus, virtual bus channels are thereby correlated with allocated busses and their protocols in the system architecture.

Group communication

Group the global communication channels which have been assigned to the same bus under the top-level channel representing the corresponding bus.

Estimate channel metrics

Annotate the channels in each bus with performance metrics estimated from the bus protocol assigned to the channels. For example, appropriate waitfor() statements are added in the communication primitives to establish execution times during simulation.

Update channel accesses

Replace channel accesses in the components with accesses to the corresponding bus interface. Update the ports and connectivity of the structural hierarchy accordingly.

Table 4.6. Architecture model guidelines.

Create high-level structure

The top-level behaviors and channels represent the components and busses of the system architecture and their connectivity corresponds to the structure of the architecture. Behaviors grouped under the top-level behaviors specify the functionality (and storage) to be implemented on the corresponding component. Similarly, channels grouped under the top-level channels represent the communication to be implemented over the corresponding system bus.

Sequentialize component functionality

The behavior hierarchy inside the component behaviors is purely sequential, i.e. there are no parallel or pipelined behavior types. Parallelism is available only at the top- level, where all the component

behaviors run concurrently.

Specify global communication

The bus channels exclusively contain abstract channels for directed communication of data values, i.e. there are no variables and random-access storage inside the bus channels.

Annotate estimated metrics

Behaviors and channels are annotated with their estimated design metrics for the components and busses to which they are mapped, respectively. For simulation purposes, appropriate delay statements are added to the behaviors and channels.

Table 4.7. Refinement rules for protocol insertion.

Insert protocol code

For each system bus, pull the corresponding protocol channel out of the protocol library.

Generate application layer

For each bus, generate the application layer that implements the abstract communication assigned to that bus, using the primitives provided by the corresponding protocol channel.

Replace bus channels

Replace the virtual bus channels in the architecture model with the hierarchical combination of application layer and bus protocol channels.

Table 4.8. Refinement rules for transducer synthesis.

Insert transducers

Insert transducer behaviors between component behaviors and bus channels that have incompatible protocols on their ports.

Encapsulate with wrappers

Replace the component behaviors with their wrapped equivalents that encapsulate the IP interface protocols.

Synthesize transducer code

Create code inside the transducer behaviors which performs a one-to-one mapping of communication primitives on one side to corresponding primitives on the other side.

Table 4.9. Refinement rules for inlining.

Inline communication methods

Move the communication functions provided by the wrappers and channels into the transducer and component behaviors where they are accessed. Variables inside the low-level protocol channels become global, shared variables. Change ports and connectivity from channel interfaces to variable accesses.

Optimize

Cancel away transducers in case the protocols on both sides of a transducer are equivalent after inlining.

Table 4.10. Communication model guidelines.

Implement bus functionality

At the top level, the behaviors which represent the components of the system architecture communicate via a set of shared variables which represent the wires of the system busses in the target architecture.

Annotate bus timing

The communication between components over their interfaces and the bus wires is modeled with accurate timing whereas the (purely sequential) behavior inside the components is at the functional level with annotated estimated delays for simulation.

Table 4.11. Implementation model guidelines.

Implement system busses

As in the communication model, component behaviors communicate over shared variables representing the wires of the system busses. Communication is modeled with accurate timing.

Implement system components

The component behaviors are replaced with a cycle-accurate model of the component implementation. Hence, computation is modeled with accurate timing, too.