A Component-Based Meta-Model and Framework in the Model Driven Toolchain C-Forge

Francisco Ortiz, Diego Alonso,
Francisca Rosique, Francisco Sánchez, J.A. Pastor

www.upct.es
www.dsie.upct.es
The C-Forge toolchain & development process

- **FraCC**: C++ framework that provides the runtime support for executing WCOMM applications.
  - Full control over concurrency of components and applications thanks to a flexible deployment easily reconfigurable (nodes, processes and threads).
  - Model transformations to schedulability analysis tool to check the real-time requirements.
  - **Key point**: Do not generate implementation code. FraCC model loader directly loads WCOMM models and interpretes them to instantiate FraCC classes.

- **WCOMM**: white-box component-based metamodel.
  - Component behaviour is expressed as Finite State Machines.
  - Admits concurrent regions inside the component.
  - The code executed by the FSM states is modeled by activities shells (wrapper of the code), with pins connected to ports and events.
  - 1 textual tool for data-types, messages and activity shells (wdam)
  - 2 graphical tools for component definition and architecture
Epsilon:

- M2M & M2T transformations,
- OCL restrictions
- Wizards.
The Robotic Platforms in Project MISSION

- C-Forge used to develop the **robotic control software** in the coordinated regional project MISSION-SICUVA.

- **Common hardware architecture**
  - Autonomous Underwater Vehicles: UPCT LVS ([www.upct.es/lvs](http://www.upct.es/lvs))
  - Test vehicle NI LabVIEW robotics Kit DaNI
  - Autonomous robotized golf cart VEGO

**Embedded PC** Nuvo-1300af-620M (Intel® Core™ i7-620M)

**PAC** - MPC5200 400-MHz processor - Wind River VxWorks real-time OS ([www.ni.com/compactrio](http://www.ni.com/compactrio/))
The C-Forge development process

WCOMM

1. Architectural draft
2. Datatypes and messages
3. Simple component definition
   i. Behaviour (FSM)
   ii. Activity shells
   iii. Code development (in parallel)
4. Instantiation of real components in the Arch. Draft.

FraCC:

A. Linking activities to code.
B. Flexible deployment (nodes, processes, threads)
Step 1: Architectural Draft

- Initial idea of an architecture: architectural draft
- The first architectural sketch is now a model
Design Process: Architectural Draft

- Draft components “empty” - only name of ports.
- It is possible to integrate already available components.
Step 2: Datatypes and messages

- Primitive datatypes already available
- Common datatypes reusable from other apps.
- New datatypes and messages through ports:
  - During the next step (3) → defining the component internals (incremental design)
Step 3: W Simple Component Definition

- WCOMM defines **white-box simple components** that encapsulate their behaviour (orthogonal regions / hierarchical states)
- Communicate by sending **messages** to each other only through their compatible **ports**
- Messages can have parameters and follow the “**asynchronous without response**” scheme

**Compared to OMG’s MARTE, WCOMM ports are **flow ports**:**
- **non-atomic** (messages can have parameters of any type)
- **bi-directional**
- **behavioural** (messages can trigger events)
Activities **encapsulate the code** to be executed in that state.

Users only model the “activity shell”:  
- the messages they **receive** (from other activities or through inports),  
- and the messages they **generate** (to other activities or to outports / or messages linked to events)  
- **Note**: the code itself (dynamic library) will be linked later in FraCC, so they can evolve independently.
Adding activities

- Textual definition of an activity shell (pins) + Graphical instantiation
Llamar atención sobre los eventos
Incremental design process

• Activities interfaces (pins) can be updated when refining the code.
• The link from activity pins to ports can also be updated.
OCL: Some included checks

- Ports have at least one message.
- Input activity pin must be connected to input messages.
- Activity pins are connected.
- Events must be triggered by messages.
- Messages are connected.
Step 4: Instantiation of components

- Coming back to the architectural draft
- Every draft component (gray) is instantiated (yellow)
- Iteratively components can be updated.
The C-Forge toolchain & development process
FraCC (A): Linking with user code.

- Generating the code (only for the activity shell)
- Linking to the user code.
FraCC (A): Linking with user code.

```c
#include "A_MisRunning.h"

//To send the messages available for this activity using

/* For using output pin OP_Stmts:
missionca_dtypes::TrackSts data;
bool error_code = push_OP_Stmts(data);
*/

/* For using output pin OP_Cmd:
missionca_dtypes::ControlSts data;
bool error_code = push_OP_Cmd(data);
*/

/* For using output pin OP_ReqPath:
missionca_dtypes::Plan data;
bool error_code = push_OP_ReqPath(data);
*/

//To receive the messages available for this activity

/* For using input pin IP_Plan:
missionca_dtypes::MissionSts data;
bool error_code = pop_IP_Plan(data);
*/

/* For using input pin IP_TrackSts:
missionca_dtypes::Path data;
bool error_code = pop_IP_TrackSts(data);
*/
```

```c
activity A_Reporting{
    IP_MissionSts << M_MissionSts;
    IP_Pose   << M_Pose;
    OP_Mission >> M_Mission;
    OP_Plan   >> M_Plan;
}
```

```c
activity A_ProcessingCmds{
    IP_Plan  << M_Plan;
    OP_Stmts >> M_Stmts;
    OP_SetPlan >> M_SetPlan;
}
```
FraCC (B): Flexible deployment
Schedulability analysis
Final: Running the application

- Framework **FraCC**, includes a **model loader** that directly interprets WCOMM models and execute applications according to the deployment model and loading the previously compiled activities code.
Conclusions

• Our goals:
  o To use models as much as possible in all steps of the development process.
  o To facilitate the integration of models with user code.

• WCOMM components are white-boxes with explicit support to define potential concurrency inside components at the model level.

• Component behaviour is modelled by FSM including an activity inside each state to model the shell of an algorithm, which will be developed apart by the user, thus allowing both (component model & code) to evolve independently.

• FraCC provides the runtime support needed for executing WCOMM applications:
  o Facilitates application deployment (number of nodes, processes, and threads), clearly separating it from application modelling
  o Control over the concurrency characteristics of the application
  o In terms of both the number of threads and their temporal characteristics.
Eclipse-based tool-chain supporting a Model-Driven and Component-Based development process

www.dsie.upct.es/cforge

francisco.ortiz@upct.es

Universidad Politécnica de Cartagena (SPAIN)