CONCURRENT OBJECT NETS - A DESIGN METHOD FOR DISTRIBUTED AND EMBEDDED REAL-TIME SYSTEMS

Wolfgang Fengler, Jürgen Nützel

Technical University Ilmenau, Faculty for Informatics and Automation, D-98684 Ilmenau, Germany, wfengler; nuetzel@theoinf.tu-ilmenau.de, http://www.theoinf.tu-ilmenau.de/ra1/

Abstract. In this paper we present an object-oriented method for the design, verification and implementation of embedded and distributed real-time systems. The method is called Concurrent Object Net (CON). The CON method is based upon extended statecharts which use specific message links for communication. For simulation and verification corresponding Petri nets are used. A platform abstraction framework for CON accesses hardware in optimized manner. An anti-slip-control of a rc-car is used to show further CON details.

Key Words. Object-orientation, simulation, Petri nets, embedded systems, real-time, anti-slip-control

1. INTRODUCTION

The design and implementation of embedded and distributed real-time systems differs hardly from traditional software design known from office applications. Despite all known differences we found object-oriented paradigm suitable for embedded systems as well. But using the object-oriented design techniques for embedded systems means to deal with two contradictory intentions. On one side the object-orientation tries to assist the designer handling huge software systems by providing techniques like abstraction, inheritance and polymorphism. On the other side designing embedded applications stands for accessing hardware directly and using provided computing resources most effectively. This often conflicts with the other side because the traditional usage of object-orientation doesn't provide the demanded code efficiency for distributed and embedded applications. For this reason why designed a new object-oriented design method directly for the design of distributed embedded real-time systems. The method is called Concurrent Object Nets. The method provides a graphical semantics for classes and objects and their interaction. An corresponding Petri net semantic of the objects supplies the designer with formal description. A hardware abstraction framework was integrated to realize optimized hardware access in combination with portability and code efficiency.

In the next section a description of the object-oriented design model is given. Afterwards the corresponding Petri nets semantic is described. In section 4 the architecture of a visual CASE tool for *Concurrent Object Net* (CON) design is introduced. Section 5 is devoted to an anti-slip-control of a rc-car.

2. DESIGN METHOD

In this section we introduce a graphical and object-oriented design method for distributed and embedded real-time systems. The method is called *Concurrent Object Nets* (CON) [3, 4] and additionally provides a framework for the abstraction of distributed embedded platforms including their actuators and sensors (see figure 1). The section starts with a discussion about different design properties which are originated from contradictory views on embedded and distributed systems.

2.1. Contradictory Views

If a software designer is directly involved with the logical view on the system to design, she normally expects three fundamental logical realization properties from a design method:

- *Concurrency*. A property coming from the environment of the real-time system. At each time different activities may run concurrently in the environment to which the system has to react simultaneously. *Concurrent Objects*: All objects (instances of Object Net classes) are working concurrently.

- *Reactivity*. A reactive system reacts continuously to signals from the environment. *Active Objects* (like actors) [5]: All instances have their own thread of control (hierarchical ones have several threads). Each instance is able to react directly to its environment.
- *Guaranty for time and logical restrictions*: In case of safety and real-time requirements facilities will be needed, which ensure that some system states will reached and some other will never occur. Additionally strict time borders for some reactions are needed too. *Safety Objects*: For every Object Net class a set of such constraints may be given. CON design ensures the fulfilment of these constraints.

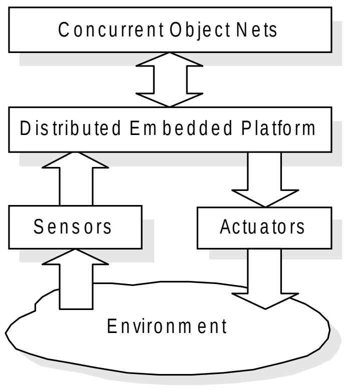


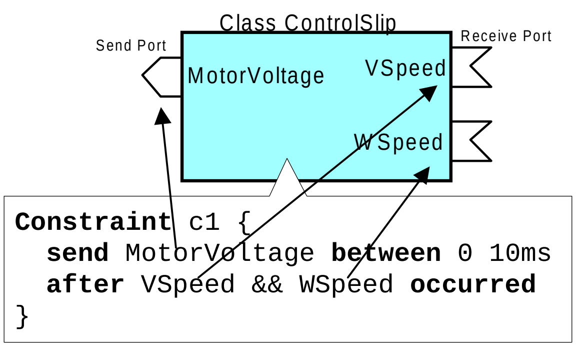
Fig. 1. The Elements of Distributed Embedded Real-Time Systems

If the designer (may be different to the one above) is involved with the physical view (implementation) on the hardware, she appreciates different properties provided by the Concurrent Object Net platform abstraction framework (PAF).

- *Portability* and *distributability*. The functionality of the specification should be validated (by simulation) before hardware details become visible. The real access function to actuator/sensor hardware should be added after the validation and verification. If an abstract actuator/sensor access exists which hides platform details, automatic software-hardware-mapping algorithms could be applied. Such algorithms generate an optimized and distributed implementation based upon the time restrictions of the specification. The platform abstraction framework (PAF) provides the designer with a class system to wrap the functionality of typical embedded controllers.
- Being *close to the hardware*. Despites the request for hardware wrapping and abstraction the implementation has to access the controller as directly as possible. The lowest classes of PAF allow method implementation in platform's native code (also in assembler).

2.2. Abstract and Refined Object Nets

To make designer's live easier when handling concurrency, safety properties and hardware access we added object-oriented features. These features allow the designer to abstract from code internals, to reuse and refine a design easily.



```
Constraint c1 {
  send MotorVoltage between 0 10ms
  after VSpeed && WSpeed occurred
}
```

Fig. 2. Abstract ON class with textual constraint

As already told Object Nets (ON) are concurrent working instances of ON classes which may be also distributed over several platform target nodes. An ON class is a kind of graphical template for the creation of Object Nets instances with certain interface and behaviour. The interface of an ON class consists of a set of so called ports. Message links connect ports of different ON instances with each other. ON instances communicate via these message links (asynchronous) by sending simple control messages or messages with user defined data structure. Beside its port interface every ON class has an internal behaviour (beside the abstract ones). Three different meta classes can be distinguished by their internal behaviour:

- An abstract ON class (AONC) has no specific behaviour. AONCs like all other ON classes may have a port interface and an optional list of time and logical constraints (see figure 2). During the design flow all instances of AONCs have to be substituted by instances with refined behaviour.
- A hierarchical ON class (HONC) encapsulates an Object Net which consists of several ON instances connected by a number of message links. These aggregated instances may also be created either from AONC, HONC or elementary ON classes. Selected ports from the interface of the aggregated instances can be exported into the interface of the surrounding HONC.
- An elementary ON class (EONC) encapsulates a hierarchical extended state machine [1] with additional time delays and time constraints. The EONC is the fundamental building block. HONCs

are only used for a better design structure. Hierarchy can be removed without change in operating semantic. The peripheral interface ports are assigned to the internal state changes. Actions with program code (which can be simulated) are assigned to the state changes. Within the actions access to actuators and sensors is possible. This is done through abstract actuator/sensor classes which couple with the ON specification of the environment. Additional local attributes (variables) extend the state space of the EONC.

Within the CON method certain inheritance rules have been defined. A subclass always refines the superclass. An AONC can be refined to either a HONC or EONC. Refining HONCs means (simplified speaking) to add new ports or ON instances (ONI). Each inherited ONI can be overwritten by an ONI which is more specific than the inherited one. EONCs can also be refined by extending their port interface. The inherited state machine can also be refined by adding new parts (like states, state changes, actions, attributes) or by refining the inherited states. The use of all inheritance rules is restricted by the CON design flow [4].

2.3. Platform Abstraction Framework (PAF)

Within the EONCs actions allow to access peripheral actuators and sensors through objects from special abstract actuator/sensor classes (ASC). These classes wrap the functionality of the hardware of actuators and sensors which are connected to the distributed embedded controller nodes. Abstract ASCs are platform independent. Their methods correspond to the complementary ASO of the Object Net environment specification which is used to create test patterns. In the case of implementation the CON platform abstraction framework (PAF) provides a technology to derive concrete ASCs from the abstract ones by inheritance. These concrete ASCs include the code for the different target controllers.

As the right half of figure 4 shows the implementation of concrete ASC methods uses two further meta classes from the PAF. *Core function unit classes* (CFUC) encapsulate the controller core specific code which is

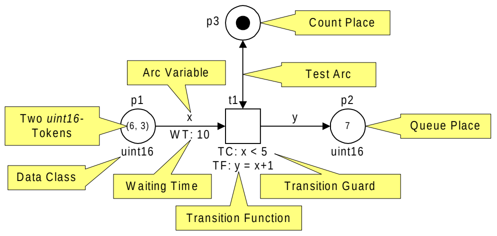


Fig. 3. High-level Petri net for the behavioural description

needed to access the different function units of the controller core. The PAF provides a selection of abstract CFUCs which reflect typical functional units like ports, watchdogs or counters. These CFUCs will be used to design platform independent *peripheral interface coupler classes* (PICC). PICCs hide the process interface around the controller core. Different DACs and ADCs are typically represented by PICCs. After the mapping of the ASOs on a specific platform CFUOs from abstract classes will be replaced by objects from concrete classes which are coded using target assembly language in an optimized manner.

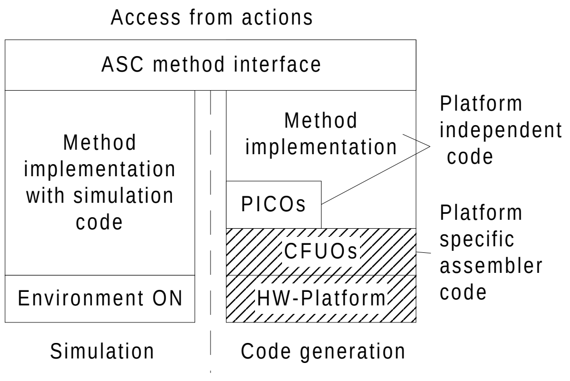


Fig. 4. Actuator/Sensor Classes for Simulation and Code Generation

Beside the ASC design PAF also supports the design of controller's communication interfaces (CI). CI classes (CIC) also use PICOs and CFUOs for their implementation. Unlike the ASCs the CIC are not visible from the Object Net specification. Their purpose is to implement the message link communication on the top of different types of protocols (e.g. CAN-Bus) and communication hardware [4].

2.4. The Design Flow with Object Nets

The design flow in the Object Net method is based on the principle of refinement through inheritance. The software designer starts to discuss the problem with the customer. As a result of this discussion the designer creates a first Object Net specification. This specification formalizes the informal requirement of the customer. It includes instances of AONCs. During the refinement (through inheritance) process the designer overwrites the instances from the first specification level with instances which are more specific.

The mechanism of refinement through inheritance is very restrictive in the CON method. Two fundamental inheritance rules define the allowed refinements. The *interface inheritance rule* is based upon the principle that the environment of a subclass can not distinguish between a subclass and its superclass if the subclass has replaced the superclass. The *constraint inheritance rule* forces a subclass to fulfil the constraints of the superclass as well.

If designer's refinements use these inheritance rules the properties of the first specification will be preserved.

An automatic (formalized) check detects whether the designer has violated the rules.

At the end of the refinement process a specification with all details will be found. This specification can be graphically simulated. The last step in the design flow transforms the fully refined Object Net specification into a specification which can be implemented on the distributed target platform.

The final mapping of the ONIs and ASOs onto the controller nodes is restricted by the node's performance parameters and the delays of the communication network between the nodes. The time constraints from the specification hold further restrictions for the mapping algorithm.

3. VERIFICATION BASED UPON HIGH-LEVEL PETRI NETS

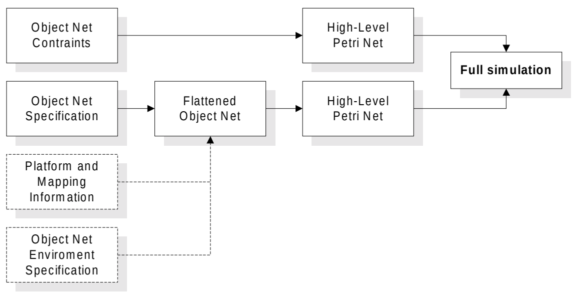


Fig. 5. Formal verification of specification constraints

For simulation and implementation every hierarchical ON will be automatically transformed into a flat ON including only EON instances coupled by message links. If an Object Net environment specification already exists it will be integrated. In the second step these flat ONs will be transformed into high-level timed Petri nets using further information from the platform (performance and delay parameters) and mapping specification. After this conversion process the resulted Petri net will be executed/checked in the attached simulator. The conversion principles are shown within the application example at the end of the paper.

The Petri net class of the simulator/checker provides several high-level features which extend traditional Petri nets (see figure 3). Tokens are able to carry data of specific structure (data class). Special queue places are used to model communication buffers directly. A time extension (waiting time) is available to describe time consumption of communication and software actions.

To check if the corresponding Petri net fulfils the timed and logical constraints of the specification a conversion is needed. After the conversion an observer Petri net will result (see figure 5). This Petri net (which includes forbidden transitions) in combinations with the Petri net form the input for the verification. If no forbidden transition fires the specification fulfils the constraints.

4. VISUAL DESIGN TOOLSET

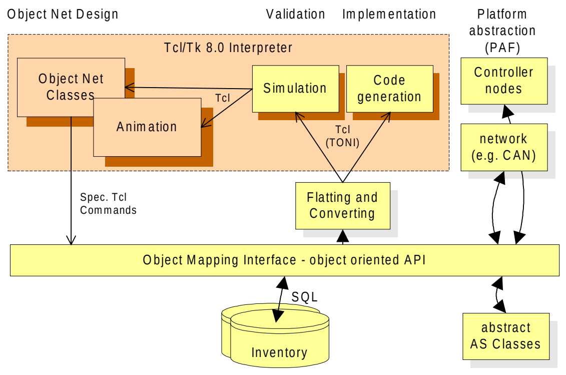


Fig. 6. The Object System Specification Inventory (OSSI)

In order to assist all the features of the CON method the modular design toolset OSSI (*Object System Specification Inventory*) was developed (see figure 6). It supports the complete design and implementation cycle within a software workgroup. OSSI provides a class/object inventory which is based on a client-server SQL database. The complete class inheritance mechanism is controlled by that inventory. It holds the complete ON class tree and all ON instances designed by a workgroup using CONs. The Object Net class browser and the platform abstraction browser allow the designer to control its project data stored in the inventory. The object mapping interface wraps the physical representation of inventory and makes the tool architecture independent from the selected database technology.

5. EXAMPLE: ANTI-SLIP-CONTROL FOR AN RC-CAR

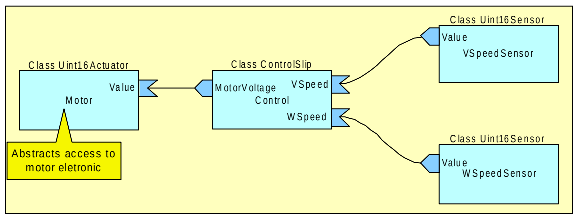


Fig. 7. CON specification of the control

We use for the demonstration of our research results [2] a rc-car (in 1:7 scale). We added to the car a distributed controller system (see figure 8). Three 16-bit controllers (C164CI from Infinion) are connected via

CAN-Bus.

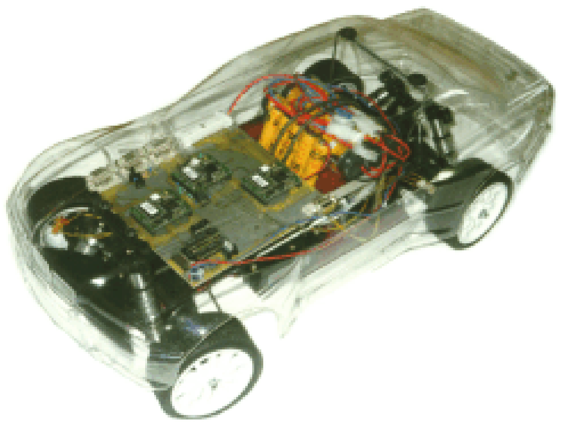


Fig. 8. BMW rc-car with three 16-bit controllers

We added an anti-slip-control feature to the rc-car, to make real-time problems visible. The car is rear-wheel driven. At the front wheel we measure the speed of the car (the vehicle speed). If the rear wheel turns faster than the front wheels the rear wheels slip. If the slip is greater than a certain amount (e.g. 1%) the driver (with the remote control) may loose control over the car.

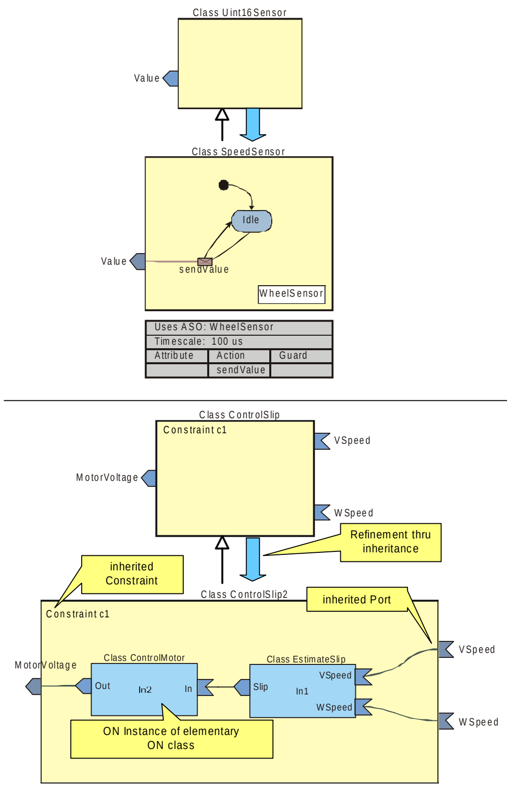


Fig. 9. The refinement thru inheritance

The task of an anti-slip-control is first to measure the speed of the rear wheel (WSpeed) and the speed of the front wheel (VSpeed). Then it calculates the slip. And after that it modifies the engine power (Motor). All this is done several hundred time a second.

In our CON example we made specific simplifications to make the design more understandable. First we removed the access of the remote control. Figure 7 shows the CON control specification. The specification consists of four ON-Instances (Motor, Control, VSpeedSensor and WSpeedsensor). On this level of design all classes (Uint16Actuator, Uint16Sensor and ControlSlip) are abstract. For the class ControlSlip some constraints (named c1) are given (see figure 2).

On a later design step all instances will be overwritten with more specific classes. The instance Control will be overwritten by an hierarchical class (see bottom of figure 9). The more specific class ControlSlip2 includes two instances of elementary type. The first one calculates the slip. The second one derives from the slip the motor voltage. The top of figure 9 shows the refinement of the instances VSpeedSensor and WSpeedSensor. The more specific class SpeedSensor is a simple state machine which calls cyclic the action sendValue. This action is written in C and uses ASC methods (see figure 4) to access the sensor hardware. In the simulation the ASC method connects the CON control specification with the CON specification of the environment.

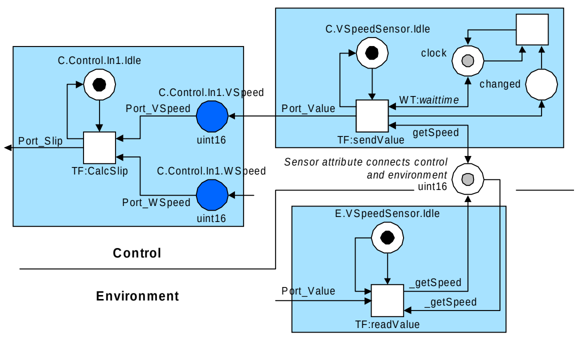


Fig. 10. A part of the corresponding Petri net for control and environment.

Before the code generation [4] starts the verification (full simulation) is possible (see figure 5). Figure 10 shows a part of the corresponding Petri net (without platform specific parameters). Two elementary instances from the control specification and on from the environment are shown.

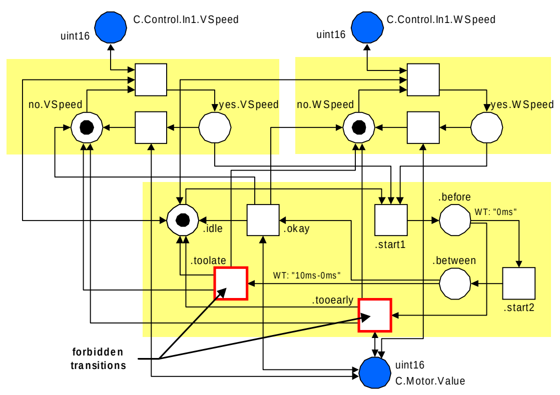


Fig. 11. The observing Petri net derived from the textual constraint

The observing Petri net which implements the constraint definition from figure 2 makes the Petri net complete for verification. The observing Petri net from figure 11 does not effect the rest of net. It is coupled via test arcs. The firing either of the transition .toolate or the transition .tooearly indicates that the real-time constraint c1 was not hold by the CON design.

6. CONCLUDING REMARKS

The Concurrent Object Net (CON) method in combination with its platform abstraction framework (PAF) was designed for easy access to the domain of distributed embedded real-time systems. Our intention was to address engineers who are familiar with hardware details but not willing to become a computer scientist. Beside easy programming of heterogenous platforms we focused on verification of safety critical systems [2]. Our future goals will be to rise the acceptance of simulation and verification in the embedded system domain.

7. REFERENCES

1. Harel, David: Statecharts: A visual formalism for complex systems, Science of Computer Programming, Vol. 8, p. 231-274, 1987
2. ISCRA Homepage, 2000: http://www.theoinf.tu-ilmenau.de/ISCRA/
3. Nützel, J.; Däne, B.; Fengler, W.: Object Nets for the Design and Verification of Distributed and Embedded Applications, EHPC´98 Orlando, USA,1998, In LNCS p. 953-962 Springer Verlag 1998
4. Nützel, J.: Objektorientierter Entwurf verteilter eingebetteter Echtzeitsysteme auf Basis höherer Petri-Netze, Dissertation, TU-Ilmenau, 1999
5. Selic, B.; Gullekson, G.; Ward, P. T.: ROOM - Real-Time Object-Oriented Modelling, John Wiley & Sons, 1994