A Component-Based Specification Approach for Embedded Systems using FDTs

Abdelaziz Guerrouat
Clausthal University of Technology
Department of Computer Science
Julius-Albert-Str. 4
D-38678 Clausthal-Zellerfeld
Phone: +49-5323-72-7172
aguerrou@in.tu-clausthal.de

Harald Richter
Clausthal University of Technology
Department of Computer Science
Julius-Albert-Str. 4
D-38678 Clausthal-Zellerfeld
Phone: +49-5323-72-7170
richter@in.tu-clausthal.de

ABSTRACT
This paper presents a framework for specification and testing of component-based embedded systems using formal description techniques (FDTs). We deal with embedded systems from the point of view of communication and thus we propose a communication model for them. We further explain the meaning of component-based embedded systems and how these can be specified using FDTs. FDTs such as Estelle and SDL are based on EFSMs (Extended finite State Machines) and have been widely used in the automation of the development process of protocols and communicating systems, i.e. for specification, analysis and validation purposes. The main goal of this work is to demonstrate the reusability of FDTs for component-based systems.

Keywords
Formal description techniques, component-based systems, embedded system, specification, testing

1. INTRODCTION
Embedded systems are becoming more and more the key technology of any kind of complex technical systems, ranging from telecommunications devices to automobiles and aircrafts. An embedded computer system is a computer system that represents a part of a larger system and performs some requirements of that system.

The growing amount and complexity of requirements on embedded systems regarding properties like safety and real-time behaviour make the software development process a costly and error-prone activity. The cost factor plays, however, a central role in today's industrial competition, for instance, between car manufacturers. The development of competitive and efficient products is imposing more and more constraints to the design of embedded systems. One of the means to reach this goal are formal methods to support the different phases of system development, i.e. specification, synthesis and validation. There are several requirements for those methods that should be among others qualities abstract, understandable, analyzable, scalable and unambiguous specification formalisms.

Component-based development represents an attractive approach in the embedded system area, in particular for the development of many variants of products [10]. While in the last few years component-based software development gained much more attention from both researchers and practitioners, testing such software systems is still however to be more studied [11]. Because one believes that once a component is sufficiently tested, it is not needed to test it again when reused. But, this is generally not true, since components may satisfy a certain application domain and fails in a new environment [11].

Formal description techniques have demonstrated their effectiveness in testing complex requirements like those for communicating systems [1] [2]. They provide a solid mean for unambiguous specification and rigorous analysis and are based on EFSMs (extended finite state machines). They differ from conventional programming languages by providing not only a formal syntax but also a formal semantic. The application of formal specifications increases the confidence in the software and the system. Especially in the area of safety-critical systems, the use of formal techniques is highly recommended [3] [8].

Testing of communication systems based on FDTs mainly concerns conformance testing. The later corresponds to a black box test. But this type of test is the most dominant and important one in component-based systems due to the nature of the constituents and properties of component-based systems. Indeed, components are independent and replaceable parts of a system and should be conform to and provide a set of interfaces. They also consist usually of special components, called COTS (commercial-off-the-shelf) that can be purchased on a component market. These are often delivered without their source code which makes other types of tests like white or grey tests less appropriate.

Actually, the mostly used formalisms to specify requirements for embedded systems are Statecharts and also UML (Statecharts are converted in UML) as semi-formal models. Although Statecharts and UML provide graphical facilities, they might lack formal and unambiguous semantics. Therefore, detecting bugs, incompleteness and inconsistencies becomes a difficult task. To alleviate these lacks many authors try to combine formal notations like Z with state-transition models [5]. Z is based on set theory and first order predicate logic and used for data structuring and abstracting. However, approaches developed around this model do not clearly address test data generation methods for analysis and validation purposes [6] [7] and/or do not deal with component-based systems.

Finite state machines are very popular in the control flow specification of state/transition-based systems and many related analysis methods have been developed [6] [7]. These support a formal test derivation which is used for validation and testing purposes. However, finite state machines lack to deal with the data flow. This shortcoming can be alleviated by using the

extended finite state machine model on which formal description techniques are based.

In this paper, we present a framework for testing component-based embedded systems by using formal description techniques (FDTs). We first identify the main components an embedded system consisting of and then show how these can be linked together to constitute the whole embedded system by using the FDT Estelle. The principle of testing such obtained embedded systems is explained, i.e. fault model, test derivation and test execution.

The rest of the paper is organized as follows. Section 2 give the basic structure of embedded systems and explains their basic communication model. The specification and testing framework for component-based embedded systems using the FDT Estelle is presented in Section 3. Finally, Section 4 concludes the paper.

2. BACKGROUNDS

2.1 Embedded System Components

An embedded system (ES) is any computer system or computing device that performs a dedicated function or is designed for use with a specific embedded software application, e.g. PDA (Personal Data Assistant), Mobile Phone, E-Book (Electronic Book), Robot, etc. That is, an embedded system is a special-purpose system built into a larger device. It is embedded as a subsystem in a larger system which may or may not be a computer system. An embedded system is typically required to meet specific requirements.

Embedded systems must usually be dependable, efficient and must meet real-time constraints. Be 'dependable' means that an embedded system must be reliable, available and safe. The efficiency mostly concerns properties like energy, code-size, run-time, weight and cost. An embedded system is dedicated for a certain application and characterized also by a dedicated user interface. Thus, knowledge about future behavior at design time can be used to minimize resources and to maximize robustness. Many embedded systems must meet real-time constraints. A real-time system must react to stimuli from the controlled object (or the operator) within the time interval dedicated by the environment.

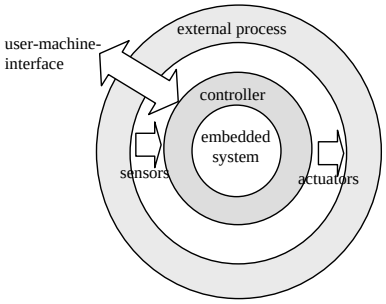

Figure 1. Main components of an embedded system

Embedded systems are frequently connected to a physical environment through sensors and actuators. They are typically reactive systems. A reactive system is in continuous interaction with its environment and executes at a pace determined by that environment. The behavior depends on input and current state for which the automata model is often most appropriate.

Figure 1 illustrates the main constituents of an embedded system comprising an *external process*, *sensors*, *actuators*, and a *controller*:

- *The external process* is a process that can be of physical, mechanical, or electrical nature.
- *Sensors* provide information about the current state of the *external process* by means of so-called *monitoring events*. They are communicated to the *controller*. For the *controller*, they represent input events. They are considered as stimuli for the *controller*.
- *The controller* must react to each received event, i.e. input event. Events originate usually from *sensors*. Depending on the received events from sensors, corresponding states of the *external process* will be determined.
- *Actuators* receive the results determined by the *controller* which are communicated to the *external process* by means of so-called *controlling events*.

The external process is usually given in advance. In contrast, the controller is often implemented by real-time hardware and software. This should allow each modification of the controller algorithm in a straightforward way each time this is needed. The controller's behavior is depending on that of the external process. The controller commands the behavior of the external process taking into consideration requirements on the process and its characteristics, such as physical laws, real time and other constraints.

2.2 Component-Based Embedded System Development

In the component-based approach for embedded systems one distinguishes a *component repository*, a *composition environment* and a *run-time environment*. The component repository consists of single specifications of the above indicated components of an embedded system: sensors, controller and actuators. These correspond to EFSMs and for each component build functional modules with respect to the used FDT. The composition environment is the embedded system specification which consists of the specification of its environment and its controller. This takes places by linking the single modules to each other by means of channels via interfaces, called interaction points (Figure 2). These modules interact with each other via broadcasting events via these interaction points. However, a sequence has to be respected in this communication. For instance, the direct communication of a module of an actuator with a sensor is not allowed. Run-time environment consists of the instantiated embedded system specification issued from the former step, i.e. the composition. Assuming the FDT Estelle, this builds a tree of linked tasks from which the system is composed. Each subtree rooted in a so-called system process or system activity task represents a subsystem [2]. The number of subsystems and the links between them are fixed once the specification is initialized.

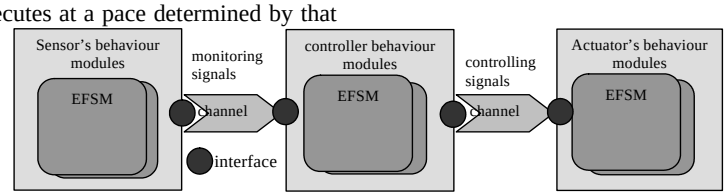

Figure 2. Overview of the composition environment for embedded system based on EFSMs

The most important component of an embedded system consists of the controller which communicates with its environment, i.e. sensors and actuators, via signals (i.e. events). To be recognized by all components, these events have to be declared as global variables for adjacent EFSMs. The output events of sensors represent input events for the controller. The events from the controller to the actuators are output events and represent input events for the actuators. They result from new computations performed by the controller that is triggered by the received input events.

Depending on the nature of sensor events (e.g. indicating the power on/of state for an electrical unit, the speed of a mobile object such as a car, etc.) the corresponding EFSM of this component is triggered and the concerned transition(s) are performed. This triggers the EFSMs of the controller whose states change. Depending on the received events, transitions in the FSMs are executed. Note, that transitions in the controller can spontaneously be triggered by other events, e.g. time out. The modeled subsequent state of the external process is computed and communicated as output events via the actuators.

To provide an intermediate specification model which better fits the behaviour part of the considered FDT, i.e. Estelle, we introduce a new EFSM, called p-EFSM (p stands for 'predicated'). This is defined as follows:

Definition 1 A *predicated extended finite state machine* (p-EFSM) is an 8-tuple $<S, C, I, P, O, T, s_0, c_0>$ where S is a non-empty set of main states, $C=dom(v_1)$ x … x $dom(v_n)$ a non-empty countable set of contexts with $v_i \in V$, V a non-empty finite set of variables, and $dom(v_i)$ a non-empty countable set referred to as the domain of v_i, P a countable set of predicates (possibly empty), I a non-empty finite set of inputs, O a non-empty finite set of outputs, $T \subseteq S$ x C x I x P x O x S x C a set of transition relations, $s_0 \in S$ the initial main state, and $c_0 \in C$ the initial context of the p-EFSM.

p-EFSM extends the conventional EFSMs for FDT mapping purposes as we will see later. p-EFSM is similar to EFSM except that in a p-EFSM the conditions on transitions are explicitly specified. This is just a notation facility and functionally and conceptually there is no difference between both models. In the rest of the paper, we indifferently address both models.

This, a transition $t \in T$ of a p-EFSM is a 7-tuple $<s, c, I, p, o, s', c'>$ where $s \in S$ is a current main state, $c \in C$ a current context, $i \in I$ an input, $p \in P$ a enabling predicate which depends on the context c, $o \in O$ an output, $s' \in S$ a next main state, and $c' \in C$ a next context.

We consider one or more p-EFSMs for each component of the system and denote them with indices *s*, *c* and *a* for *sensors*, *controller*, and *actuators*.

Interdependencies between these components are described as follows:

- Let be given a transition
  $t_s \in T_s$: $t_s = <s_s, c_s, i_s, p_s, o_s, s'_s, c'_s>$ with
  $s_s \in S_s$, $c_s \in C_s$, $i_s \in I_s$, $p_s \in P_s$, $o_s \in O_s$, $s'_s \in S_s$, $c'_s \in C_s \Rightarrow \exists\ t_c \in T_c$
  $| \ o_s \equiv i_c$
  That is, each output event generated by sensors must trigger a transition of the controller. This event represents an input event for the triggered transition. We assume here that the predicates related to the transitions are satisfied by the actual context.
- Let be given a transition $t_c \in T_c$ with $s_c \in S_c$, $c_c \in C_c$, $i_c \in I_c$, $p_c \in P_c$, $o_c \in O_c$, $s'_c \in S_c$, $c'_c \in C_c$,
  if $i_c \in O_s \Rightarrow \exists\ t_s \in T_s$ and $i_c \equiv o_s$.
  This means that if there exists a transition of the controller whose input event belongs to the set of output events of the sensors then it must exist a transition of the sensors whose output event is identified with the given event.
- Let be given a transition $t_a \in T_a$: $t_a = <s_a, c_a, i_a, p_a, o_a, s'_a, c'_a>$ with $s_a \in S_a$, $c_a \in C_a$, $i_a \in I_a$, $p_a \in P_a$, $o_a \in O_a$, $s'_a \in S_a$, $c'_a \in C_a \Rightarrow \exists\ t_c \in T_c$: $t_c = <s_c, c_c, i_c, p_c, o_c, s'_c, c'_c>$ and $o_c \equiv i_a$.
  Each transition of actuators must be only triggered by the controller and must match the output event of the triggering transition of the controller.

Estelle is a standardized formal description technique (International Standard ISO 9074) based on concepts of structured communicating extended state automata and Pascal. It is oriented towards the specification of complex distributed systems, in particular communicating systems. A specified system is presented as a tree of tasks where each task has a fixed number of input/output access points (interaction points). Within a specified system it exists a fixed structure of subsystems (sub-trees of tasks) and communication links between subsystems.

SDL (Specification and Description Language) is an object-oriented, formal language defined by The International Telecommunications Union Telecommunications Standardization Sector (ITU) (formerly Comité Consultatif International Télégraphique et Téléphonique [CCITT]) as recommendation Z.100. The language is intended for the specification of complex event-driven real-time, and interactive applications involving many concurrent activities that communicate using discrete signals.

Indeed, EFSMs can functionally describe system components that may be blocks or modules depending on the used formal description technique[1].

3. SPECIFICATION AND TESTING BASED ON *Estelle*

3.1 Specification

A specified embedded system is a tree of tasks (p-EFSMs) which can be categorized in three classes corresponding to controller, sensor and actuator modules. They are organized in an hierarchical structure (parent-son-relationship). Each task has a fixed number of Input/Output access points (interaction points) which can be associated to controller, sensors or actuator modules. Bidirectional communication links may exist between tasks (between their interaction points). Within a specified embedded system exists a fixed structure of subsystems (sub-trees of tasks), corresponding to controller, sensors or actuators, and of communications links (between them) (s. Figure 3). Within a subsystem both structures (of tasks and communication links) may change dynamically. Tasks exchange interactions:

[1] SDL uses the 'block' concept whereas Estelle 'module'.

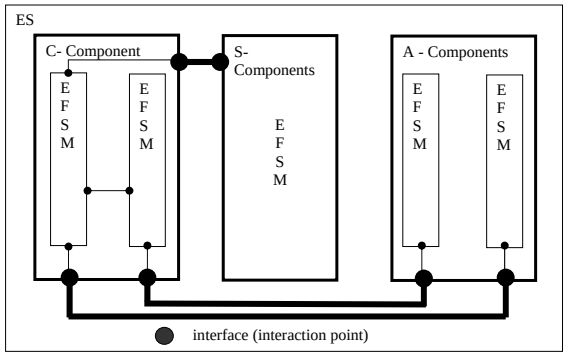

Figure 3. Structuring and communication in an ES-component-based specification

- A task may send an interaction through its interaction point to a task linked to it, e.g. from C to A via the interaction points in C and A which are linked to each other (Figure 3).
- An interaction received by a task, as its interaction points, is appended to a FIFO queue associated to this interaction point. A FIFO queue may be either associated to one interaction point (individual queue) or to many interaction points (common queue)

A task may export variables towards its parent which can access them (read and write).

Parallelism: Two kinds of parallelism can be expressed in a ES specification:

- Asynchronous parallelism: only between (actions of) tasks of different subsystems
- Synchronous parallelism: only between different (actions of) tasks of the same subsystem. Synchronous parallelism between actions means that all actions have to complete their parallel execution before other actions can be executed in parallel.

Time notion:

- Execution time of tasks (actions) is assumed unknown because it is implementation dependent.
- Some actions can be specified in such a way that their execution will be delayed. There are two delay values (min and max) which may be specified. The values of these delays are supposed to be modified by an independent process.

3.2 Syntax Overview

3.2.1 Channel Definition

```
channel NAME (IP1, IP2)
  by IP1:
       interactionName1 (typed parameters);
       …;
  by IP2:
       interactionName1 (typed parameters);
       …;
  by IP1, IP2:
       interactionName1 (typed parameters);
       …;
```

A *channel* determines two channel types: Channel_NAME (IP1) type and Channel_NAME (IP2) type. A *channel type* defines two sets of interactions: those which can be sent through an interaction point (interface) of this type and those which can be received through an interaction point of this type (Figure 4).

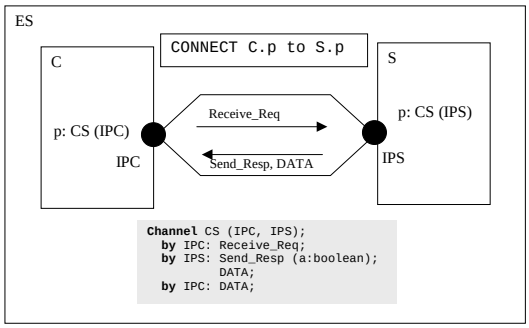

Figure 4. Interactions via a channel in an ES-component-based specification

3.2.2 Module Definition

The module definition consists of three main parts: declaration part, initialization part and transition part. The declaration part defines the manipulated objects like constants, types, variables, functions or procedures, state, stateset, channels, (sub) module headers and bodies, module variables etc. The initialization part initializes and controls variables and state variables, creates subtasks (sub-module instances) and establishes communication links. The transition part is the most important because it specifies the embedded system behaviour.

3.2.3 Transition Part

The transition part is composed of a set of transitions. Each transition has two parts: conditions and actions. Conditions are formed by the following clauses: `when, from, provided, delay, priority.` The actions are defined by the clause `TO` and PASCAL-Program with some extensions and restrictions:

```
WHEN clause
       when
interaction_point_id.interaction_name
FROM clause
       from state
       from stateset
PROVIDED clause
       provided Boolean expression
DELAY clause
       delay (integer_expression)
       delay (integer_expr1, integer_expr2)
TO clause
       to state
       to same
       output
```

It is easy to map a p-EFSM specification onto the behavioral part (transition part) of an Estelle module. The *when clause* corresponds to input events in p-EFSM, *from* to edge state, *provided* to predicate, *delay* is a timing special input event, *to* to the tail state and *output* to output event.

If the formal specification is provided in form of p-EFSMs (corresponding to a module) or in FDT many properties (completeness, correctness, consistency, safety, reachability etc.) of an embedded system can be automatically checked. In addition, different phases of the development process (analysis, implementation derivation, test data generation, diagnosis) can be unambiguisly and effectively supported [6] [7] [9].

3.3 Test Generation Methods

There exist many test generation methods that are based on FSMs and which can be under some assumptions adapted to

EFSMs [6] [7] [8]. Some of them are able to detect only certain errors classes, whereas other allow to cover all errors classes. All these methods based on FSMs have a common basic idea. A test sequence is a preferably short sequence of consecutive transitions that contains every transition of the FSM at least once and allows to check whether every transition is implemented as defined. To test a transition, one has to apply the input for the transition in the starting state of the transition, to check whether the correct output occurs, and to check whether the correct next state has been reached after the transition. Checking the next state might be omitted (transition tour method) or be carried out by means of distinguishing sequences (checking experiments method), characterizing sequences (W-method), or unique input/output sequences (UIO methods). Some of these methods were also extended to nondeterministic FSMs [7].

4. CONCLUSION

In this paper we presented an approach based on formal description techniques for specification of component-based embedded systems. The intermediate model EFSM allows to specify a system component independently of a given FDT, however, easily translatable in a preferred FDT, i.e. Estelle or SDL. We described the basic structure of embedded systems and demonstrate how a component-based approach can be applied for them using Estelle as FDT example. The main goal is to reuse the many well-known methods (automatic analysis, test data generation, validation, diagnosis, formal fault models) that have been since decades developed around state/transition-based models because formal approaches are very recommended in the today's growing complexity of embedded system requirements, especially regarding safety real-time property.

In a future work, we plan to specify a real-life embedded system from the automotive area by using FDTs.

REFERENCES

[1] Specification and Description Language SDL '92. ITU-T Recommendation Z.100, 1992.

[2] Information processing systems – Open Systems Interconnection – *Estelle: A formal description technique based on an extended state transition model.* International Standard ISO 9074, 1989.

[3] Buessow, R., Geisler, R. and Klar, M. Specifying safety-critical embedded systems with statecharts and Z: A case study. *In Proceedings of Fundamental Approaches to Software Engineering (FASE'98)*, Lisbon, 1998.

[4] Mendler, M. and Luettgen, G. Statecharts: From Visual Syntax to Model-Theoretic Semantics. *In K. Bauknecht, W. Brauer, and Th. Mück (editors), Workshop on Integrating Diagrammatic and Formal Specification Techniques (IDFST 2001)*, pages 615-621, Vienna, 2001.

[5] Potter, B., Sinclair, J. and Till, D. Introduction to Formal Specification and Z (2nd Ed.). Prentice Hall PTR; 1996.

[6] Aho, A. V. et al. An optimisation technique for protocol conformance test generation based on UIO sequences and Rural Chinese Postman Tours. *In S. Aggarwal and K. Sabnani, editors, Protocol Specification, Testing, and Verification*, New Jersey, 1988.

[7] Fujiwara, S. et al. Test selection based on finite state models. *IEEE transaction on Software Engineering* 17(6): 591-603, 1991.

[8] Richter, H., et al. A Concept For a Reliable, Cost-Effective, Real-Time Local-Area Network for Automobiles. *In Proceedings of Joint conference Embedded in Munich and Embedded Systems*, Munich, 2004.

[9] Henniger, O., Ulrich, A. and König, H. Transformation of Estelle modules aiming at test case derivation. *In A. Cavalli and S. Budkowski (eds.), 8th International Workshop on Protocol Test systems*, Chapmann & Hall, 1995.

[10] Crnkovic, I. Component-based approach for embedded systems. *Ninth International Workshop on Component-Oriented Programming*, Oslo, 2000.

[11] Beydeda, S. and Gruhn, V. Testing Component-Based Systems Using FSMs. In Beydeda and Gruhn (Eds.), Springer-Verlag, 263-280, 2004.