Esterel and Java in an Object-oriented Modelling and Simulation framework for Heterogeneous Software and Hardware systems. The SEP approach.

Frédéric Mallet
Laboratoire I3S UPRES_A 6070 du CNRS
41 Bd Napoléon III
06041 Nice cedex France
Frederic.Mallet@unice.fr

Fernand Boéri
Laboratoire I3S UPRES_A 6070 du CNRS
41 Bd Napoléon III
06041 Nice cedex France
Fernand.Boeri@unice.fr

Abstract

The size of Today's digital systems is increasing very quickly. So design tools have to allow the maximum reusability and an adapted level of description depending on our goals during each part of design cycle. Moreover, systems require more and more heterogeneous competency domain. Then we have to be able to manage the integration of complex and heterogeneous software and hardware systems. In some previous articles, we presented an object-oriented method and the related tool, which were demonstrated to be useful in order to model and simulate hardware digital architectures and their software applications in order to obtain performance estimations. This paper firstly intends to show the easiness to integrate in our framework some capabilities to describe parts of system behaviour with other formalisms. Indeed, due to description power of our generic object-oriented model and without any modification, we managed to take care of components the behaviour of which is described using the synchronous reactive language Esterel. And secondly, we illustrate the use of our new extensions to model efficiently an automatic control system for sprinkler.

1. Introduction.

1.1. Our objectives.

The size of Today's digital systems is increasing very quickly. So design tools have to allow the maximum reusability and an adapted level of description depending on our goals during each part of the design cycle. These tools must allow designers to estimate performances as soon as possible in order to reduce time-to-market and design cost. So reusability of already validated components is definitively a capital point for the design tools. That is for why our objectives are to develop a simulation environment using high-level software models of digital software and hardware architectures. Since the complete formal proof of such architecture is not possible, the simulation is the only way to make some basic verification and estimate performances. And our object-oriented modelling method induces flexibility and reusability furnishing a component-based programming framework. Since components are designed in an independent way, they are not linked to any other components and can then be used in any other projects.

Most of classical modelling techniques use low-level capacities of hardware description languages such as VHDL and VERILOG. But most of these languages are strong typed and mainly designed to synthesise hardware architectures, then they use too much specific and useless information in first time of the design flow. They do not include object-oriented capabilities even if some propositions were made in this way: OO-VHDL [12] [3]. Meanwhile, the extension to object orientation from strong typed imperative language may cause some analysis problems and results in just a complex language.
We already proposed an object oriented technique [10] and the related tool [11] designed to allow an incremental modelling with an abstraction level adapted to specific needs of the designer as well as a good flexibility and reusability for already validated components. Lots of parameters often needed with traditional approaches are just ignored by our model such as component families, propagation delays or specific types. This objective was attempted by the definition of an object-oriented generic model. This model only depends on general features about targeted architectures. The chapter 2 introduces the generic model interest and presents an overview of its powerful capability.

In addition to reusability, systems require more and more heterogeneous competency domains. Then we have to manage the integration of complex and heterogeneous software and hardware systems. Our previously introduced method and tool were demonstrated to be useful in order to model and simulate hardware digital architectures, their software applications and obtain performance estimations. The chapter 3 intends to show the easiness to integrate in our framework some capabilities to describe parts of system behaviour with other formalisms than our own formalism construct upon the Java language. Indeed, due to the description power of our generic object oriented model and without any modification, we managed to take care of components the behaviour of which is described using the synchronous reactive language Esterel [4]. This constitutes an important step because research teams always fight on the respective interests of control-based language and data-based language. This marks the difference between the reactive and control behaviours and the transformational and the computational behaviours. The addition of the synchronous reactive model of computation in our Java oriented framework continue series of propositions [1, 5, 6, 2] to integrate both approaches for the description of heterogeneous systems. Indeed, specific languages in both domains are very efficient but we need some tools to combine efficiently several formalisms and keep their specificity and properties. Some important projects propose actually some integrated environments. The paragraph 1.2 exposes an overview of their features and differences with our approach.

And finally, we illustrate in the chapter 4 the use of our new extensions to efficiently model an automatic control system for sprinkler.

1.2. Other approaches and products.

POLIS is a co-design environment for software/hardware embedded systems. It is based on a model of computation called CFSMs (Co-design Finite State Machine) and is proposed by the Berkeley University, California. This model constitutes a network of synchronous tasks connected using asynchronous communication protocols. By now, the POLIS research has focused on control-oriented embedded systems. Their very first objectives were to generate by construction an hardware/software implementation from a functional formal description. This design flow should allow designers to obtain accurate performance estimations. Because of the huge size of systems, formal proof is not possible. Since the communication delays are not bounded, time constraints cannot be guaranteed. Then, the only way to obtain some estimation results is the simulation. To perform co-simulation POLIS uses Ptolemy, which is a co-simulation environment for heterogeneous architectures. Ptolemy is supposed to allow the integration of several models of computation in order to perform a heterogeneous simulation. With that goal, users are supposed to implement some wormholes between different interesting domains. For an external user, that is not so easy and the integration of synchronous programming (esterel) in CFSMs has just been done recently [5].

Some commercial tools like Cardtools systems (computer aided real-time design tools) and Coware tool are graphical co-design and co-simulation tools, which allow the entire system description and achieve some very interesting simulation features. But since the task behaviour code is written using an imperative language (VERILOG, C or their own description language: Task Behaviour Model), the reusability is not optimum. This is because particularly in the Task Behavioural Model code components use some external references by example to shared memories or global signals, and that is generally essential with imperative languages. Then, such components can only be used in another framework using identical shared memory.

With our method, we want to show that object-oriented and component-based techniques may be useful for modelling and simulating digital architectures. This model demonstrates good reusability and flexibility properties and then allows us some easy extension toward the heterogeneous modelling of hardware and software architectures. Then we are now able to model components using the synchronous reactive language esterel. This process was simplified due to the closed relationship between the constructive electrical model and digital architectures. The knowledge of the internal process is not required for all users; it is just required for user who wants to add some models of computation. Then, other users just have to use inheritance property to describe autonomous components.

Recently, the Berkeley University has done a similar component-oriented description with the Ptolemy2 project written in Java.

2. Why use a generic object-oriented model?

2.1. Benefits of object-orientation

Object-oriented languages were born since developers want to design software around data structures more than around functionality. Then objects are supposed to encapsulate into a same structure every tool necessary to manage a specific data structure. So, they contain the data information and all specific actions about these data (methods).

For hardware design, it seems very natural to define a class of object, which will represent a hardware component. In SEP, each instance of this class is able to manage a set of services, which constitutes its basic behaviour and a set of ports, which constitutes its interface of communication with other components. In the same way, ports may be defined as a specific class of objects. These objects will be dedicated to manage communications with other components and communications to their related component.

Then, each component constitutes a processing structure and a base of communication with other completely autonomous components.

Moreover, in this way data structures can get more and more complexes and completed by inheritance. It is easy to say that a data structure is the specialisation of another one. By example, we can say that a shift register is a specialisation of a basic register with a serial input, a serial output and a shift operation. We can also say that a RAM is a specialisation of a ROM with an output bus and a read operation. This specialisation is called inheritance and may allow designers to reuse some existing data structures and specialise them into a more adapted data structure. Specialised data structures are called sub-classes of their super classes.

With the inheritance concept comes the polymorphism concept. This allows redefining methods of super classes into their sub-classes. Then, the choice to execute the right method is done because of the dynamic identification of the concerned object type. The load method of the basic register can be overloaded to process the serial output in addition to the parallel output.

You can find more complete information about objects in specialised books [13]. This paragraph only intends to show some points to encourage more people among hardware concerned designers to use object-orientation.

2.2. Model capability overview.

Our entire graphical framework is based on its capability to manage components and links described by our generic object-oriented model. By generic we mean that the model construction is not on dependency of some architecture specific features. So we should be able to model every architecture types (RISC, DSP, CISC, VLIW) as well as control systems, multiprocessors or distributed architectures. In fact, it consists in the definition of the minimal communication interface between different autonomous components. This interface imposes some rules for event propagation and data transfers independently with the operating system or programming language. This generic model allows the internal description of basic components as well as fitting together existing components connected by communication lines (control signals, data busses). Then, each component is designed independently from the others and becomes definitively reusable. Then, the difficulty consists in guarantying that component description is not ambiguous and is not dependent on fitting.

The generic model core is entirely described in [10] using the Unified Modelling Language [14]. The graphical framework features and its capability to use the model are introduced in [11] as well as the description of components in library.

The model describes every component represented by the 'materialComponent' class with their features and connection properties. A 'materialComponent' manage a set of ports and a set of services. Each port has a specific sensitivity associated to a service. The entire structure is able to perform behaviour as shown by the Figure 1.

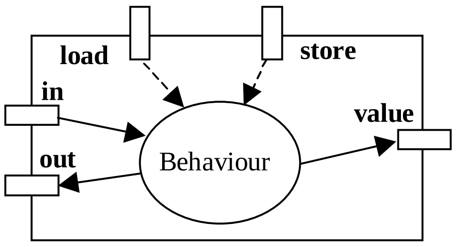

Figure 1 - The materialComponent model

Then, we add to the model some methods to describe the behaviour of each component. This can be done using two different ways. The first solution is the structural representation by fitting together some components and making some links (see Figure 2).

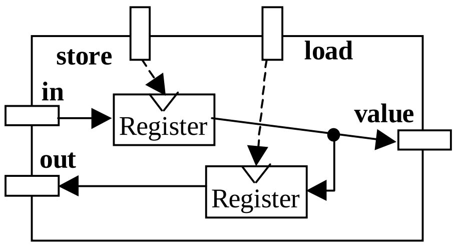

Figure 2 – The load/store register: The structural description

The second possibility is the use of a description language. By now, the description language used is the Java language (see Example 1).

This article intends to show the ability to use new description language eventually corresponding to other models of computation like the esterel language (see chapter 3). A VHDL or VERILOG description might easily be integrated but it does not appear that this could have an interest by now excepting the possibility to use existing VHDL or VERILOG code. The definition of an abstract class, which implements each necessary conversion and eventually a new scheduling method in accordance with the default one, may perform such integration.

The Example 1 presents a software description in Java, the syntax of which is fully explained in previous articles.

```
package component.basic;
public class LSRegister extends
          modele.simulation.EdgeComponent {
  public LSRegister() {
    addInactivePort ("in", LEFT);
    addInactivePort ("out", LEFT);
    addRisingEdgePort("load",TOP).setService("load");
    addRisingEdgePort("store",TOP).setService("store");
  }
  public void load() {
      setPriority(1);
      emit("out", read ("value"));
  }
  public void store() {
      setPriority(0);
      emit("value", read ("in"));
  }
}
```

Example 1 – Load/Store register: Java description.

2.3. Communication and port sensitivities.

Components are designed in a completely autonomous way and then any kind of connections are allowed. But it is clear that some connections should be forbidden. In order to resolve this problem, a technique assert-based could be implemented easily [8]. Such a technique may allow a type validation according to developer prerequisites and is about to be searched forward.

The communication type used is data-driven like. Different components may understand the receipt of same data flows in different ways. The receiver always decides alone about the use of data from external components. This behaviour is a dependency of sensitivity associated to communication ports. This sensitivity engages the reaction associated and the execution of corresponding services (methods). Each service activation is seen as an event and pushed into a queue until the scheduler decides to active it. Relatively to different sensitivities we are able to distinguish three different kinds of components. Modele.simulation.EdgeComponent represents sequential components and should declare some edge sensitive ports. These component services generally have constraints about their relative execution in logical time. The relative execution of several services by a same component and relatively to execution of other services by other components must not depend on scheduling strategy or operating system implementation. Combinatory components called modele.simulation.LevelComponent should declare level sensitive ports. Whatever the relative execution order of different services is, the result is supposed to be identical. Meanwhile, behaviour may differ on transitory results. In a combinatory system, output values only depend on input values. Finally, time sensitive components represent components with continuous behaviour and do not have any sensitive ports. They are called modele.simulation.TimeComponent. These components react spontaneously to the spending time: by example a pomp with defined throughput or a clock with known frequency. They only send data in correspondence with the spent time. These components must declare their time sensitivity to the scheduler and to the time manager. The scheduler defines a coherent temporal policy for each of these components and calls their services at the right time.

Using object introspection properties, each port is able to take the entire responsibility about its sensitivity and launch the reaction into its associated component. For this purpose, name of the associated service is given to each sensitive port by a string value as shown by the following example:

```
addRisingEdgePort("rst", TOP).setService("reset");
```

In this example, an EdgeComponent declares a rising edge sensitive port called ‘rst’ with a service called ‘reset’. When the data sequence low-level value followed by high-level value occurs, the port has to schedule a reaction with the execution of the ‘reset’ method of its related component. Obviously a default service name is defined: 'report'.

There are five different kinds of ports: insensitive ports (InactivePort), edge-sensitive ports: rising or falling (EdgePort), level-sensitive ports (LevelPort), always-sensitive ports (VirtualPort) and service-sensitive ports (ServicePort). The VirtualPorts represent ports of containers, they are used for the structural behaviour description and they only communicate data between the outside and the inside of components. ServicePorts are used to multiplex entries in a LevelComponent. An example to illustrate the use of ServicePorts is presented in paragraph 3.3. By now, no more sensitivity type seems to be necessary. A meta-model of the port behaviour should be proposed in order to model valid sensitivities.

3. The integration of the synchronous reactive model of computation and Esterel.

3.1. The Strl component.

The Strl component reacts synchronously with the occurrence of a rising edge on its tick port. Each tick identifies the beginning of the new synchronous instant. The implementation in our framework of components the behaviour of which is described using the synchronous reactive language esterel, was performed by the description of the **component.basic.Strl** class. So by inheriting from this class, an object is able to react identically to an esterel description. Each of these components has some inputs and outputs. Moreover, it owns a particular asynchronous reset input and two outputs representing the instant end (halt) and the synchronous instruction termination (ret) as shown by Figure 3.

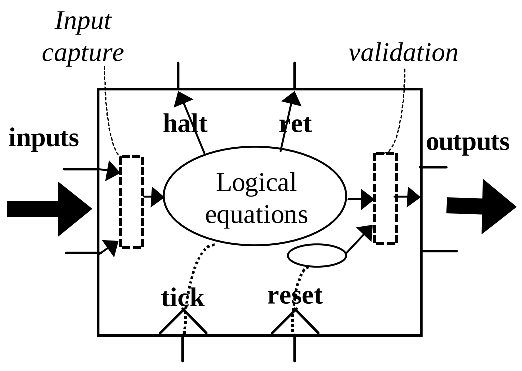

Figure 3 – The Strl component

In fact, the Strl component constitutes an execution machine [7] for an esterel program.

In order to conserve good properties established using esterel development tools about formal proofs, we did not write an esterel compiler but we directly use equations generated by the esterel compiler. There are different output file formats usable, we first chose to implement the Ssc and the Blif format. We chose the Blif format because of its simplicity and the Ssc format because of its completeness. The generated equations can be decomposed into four main types:

- The **logical** equations**:** Or, And, Not.
- The **sequential** equations made of latches and representing the time.
- The **arithmetical** equations: addition, subtraction and comparison.
- The **'jump'** equations which represents calls to external tasks.

The two last equation types (arithmetical and jump) are not allowed by the Blif format. Indeed, this format is generated only if esterel program is pure that is to say without neither any valued signal nor any task.

Another possible classification is to consider separately combinatory equations (arithmetical, logical and jump) and sequential equations. The latter group concretises sequence between two synchronous instants and constitutes the capture of inputs for the combinatory part. This capture is supposed to represent the stability of the system. These sequential equations are synchronised by the tick signal. This second classification is illustrated by the Figure 4. The instantaneous reaction of synchronous systems refers to a reaction between two consecutive ticks.

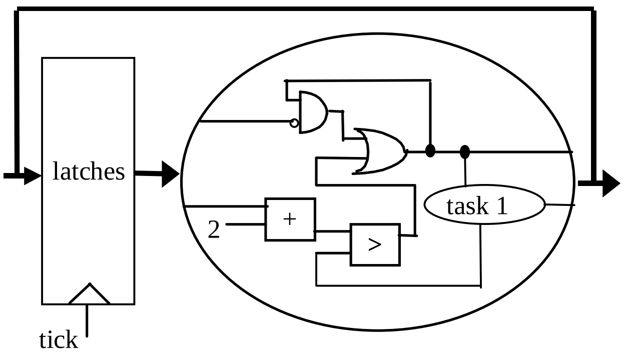

Figure 4 – Logical equations

3.2. The constructive electrical model and our implementation.

The constructive electrical model for esterel program proves by construction the conservation of semantic properties by compilation from esterel code to logical equations. Nevertheless, lots of difficult points have to be solved by the execution machine. The task management and discretisation from continuous or asynchronous processes have to be treated by the execution machine.

The Strl component core composed of logical equations is modelled by the abstract class **modele.sync.Archi**. The Archi object constitutes a sorted list of logical equations (interface **modele.sync.Equation**). These equations define values of output and intermediate signals depending on input and intermediate signals. Each input and output are associated to a Strl component input or output. Each of these inputs is captured at the beginning of the instant. An Archi evaluation consists in the one-time evaluation of each registered equation in the specified order. The esterel compiler calculates this order after analysis of dependencies.

When the designer wants to build an Strl component, he has to specify the associated Ssc or Blif file, this file is read respectively to the associated modele.sync.ArchiSsc or modele.sync.ArchiBlif which both implement the modele.sync.Archi abstract class. The abstract modele.sync.ArchiAdapter class does the common part of the parsing work, other actions are automatically done using the polymorphism property.

3.3. Taking care of esterel tasks.

Some of the previously introduced logical equations consist in the execution of actions about esterel tasks. These tasks are controlled using five specific control signals (start, activate, return, kill, suspend). Moreover, they sometimes have input or output parameters associated to intermediate signals. Each of these parameters is transmitted to the task during the ‘start’ operation. Output parameters are updated during the ‘return’ operation.

Tasks were introduced in the esterel language in order to allow the asynchronous execution of code independently with the synchronous esterel instant. The task termination is mentioned to the synchronous module by the ‘return’ signal occurrence. This signal is emitted by the task.

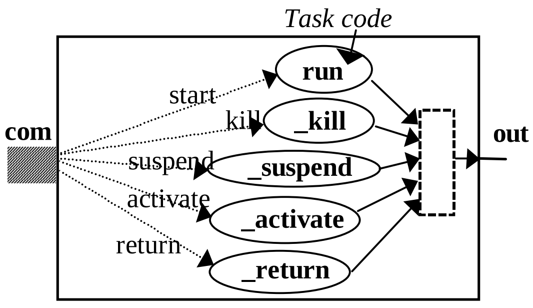

Figure 5 - Task Strl

With our proposed model, the implementation of an esterel task can be done by inheritance from the **modele.sync.TaskStrl** class. This class owns a service port called ‘com’. This port receives string value (modele.simulation.StringValue) corresponding to control value for esterel tasks and calls associated methods (run, _return, _kill, _suspend and _activate) as shown by the Figure 5. Then, user just needs to overwrite one or more of these methods. The modele.sync.TaskStrl extends the **java.lang.Runnable** class, so it can be associated to a Thread and then its execution is completely independent from the Strl component execution. And finally, this class extends the modele.simulation.materialComponent class and then can be inserted in our Java framework in the same way as any of the other continuous, asynchronous or synchronous components and independently from a component.basic.Strl component. Let us note that every task is able to emit data throw output ports to other classical material components.

In some rare cases, the task has to synchronise with the esterel environment. This synchronisation can be performed using the waitTick method which spies the activate signal in order to wait for the next esterel instant.

The quick and easy integration of the esterel language in our model without any modifications is due to the use of a generic object-oriented model and component-based simulation method.

4. An application to an automatic control system for sprinkler.

4.1. The sprinkler overview.

This system was introduced before in [9], and constitutes an automatic control system for a sprinkler. This sprinkler behaviour is highly reactive and can be decomposed into two parallel actions:

1. The mixing module: Mixing water and in the chemical fertiliser in order to obtain the right concentration called reference concentration. The reference concentration depends on the type of the watered plants and is given dynamically by the user (Param task). The main constraint is to always get a minimum quantity of liquid, but not too much.
2. The watering module: Watering the diluted fertiliser using pomp and respecting the right humidity for watered plants (reference humidity). The reference humidity is also given dynamically by the user (Param task).

Both references constitute parameters of an external task. This task has to access a database. Moreover, the sprinkler has to report the overall quantity of fertiliser used in order to get the right humidity for the watered plants. This is done by the external ‘report’ task.

The Figure 6 introduces an overview of the specified system. On this figure, bold names represent signals for esterel system (input or output) and needed sensors to extract the information.

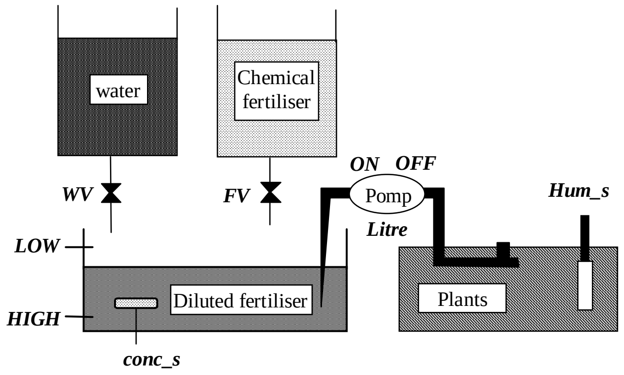

Figure 6 – Automatic sprinkler

4.2. Identifying problems.

This simple problem hides some interesting common difficulties. It includes several concurrent activities: supervising diluted fertiliser level and concentration registering information about consumption and tap handling. This is a reactive behaviour and then a reactive synchronous language like esterel is well adapted to describe such behaviour.

This problem also includes a computing activity because of their access to a database system to extract reference parameters and to register consumption results. An imperative or object-oriented language (C or Java) can easily perform this. These accesses have to be controlled by esterel environment then the use of esterel tasks seems to be the best choice.

Finally, pomp, recipients and taps have continuous behaviours. They do not react to synchronous commands but they react with continuous variation. Their behaviour is on dependency of the absolute time. So this problem requires a simulation environment allowing an heterogeneous description just like synchronous discrete, data-oriented and continuous.

4.3. The proposed solution.

Due to generic properties of our model, we can model each of these components to have a complete overview of the system behaviour. We described the controller behaviour with the esterel language. In fact, we just used the code from [9]. This code can be validated and some properties can be proved using the esterel development environment. We wrote with the Java language the task behaviour. An access to database can be quickly implemented with the Java language because of the high-level application programming interface. It is very easy to access files or common databases eventually on a distributed system using the Java Data Base Connection (JDBC). JDBC permits concurrent accesses to an enterprise's distributed database. The esterel tick does not schedule the reaction of these tasks, so other components are not waiting for the data base connections and interactions (see paragraph 3.3).

Finally, models of continuous components just like recipients and pomp are made using inheritance from the TimeComponent class. User defines the pomp and taps throughput and a clock frequency associated to the esterel tick. Our first steps were to define new components to add to our component library. These components are a recipient and a tap. Then we add to this type a field for concentration and we overwrote the arithmetic operations to take into account these parameters. Then we decided to add a liquid type inheriting from the Value class to model a liquid. A value of this type represents a volume of liquid, so it is quantified using the litre unit. Anyone can easily add some new parameters to a liquid using inheritance.

The recipient interface is presented by Figure 7. It has a level sensitive input port named 'control'. This port intends to receive liquid value, specifying liquid quantity to add or remove. Then, this component is able to export to external components some data about its state. It exports full and empty binary signals. And it exports a quantity signal supporting a liquid valid that contains the actual volume and concentration of component.

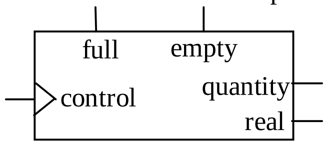

Figure 7 - The recipient component.

Finally, the 'real' output port shows the last transaction done in the recipient. The tap (see Figure 8) is a TimeComponent, its time sensitivity defines its throughput. The 'on' and 'off' input ports enable and disable the component's time sensitivity. Then, a liquid value representing quantity of liquid moving throw the tap is sent across the output port 'out'. The entire system was modelled and simulated using those components as shown by Figure 9.

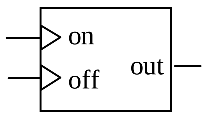

Figure 8 – The tap component.

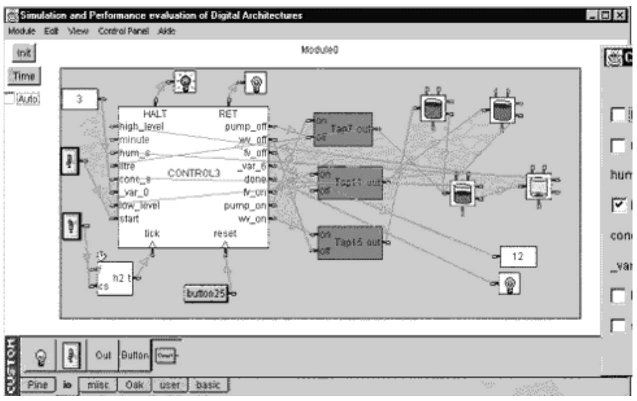
Simulation and Performance evaluation of Digital Architectures
Module Edit View Control Panel Aide
Init
Time
Auto
Module0
HALT
RET
CONTROL3
tick
reset
button25
Out
Button
Pine
misc
Oak
user
basic

Figure 9 – Sprinkler system in SEP framework and esterel control panel.

5. Conclusion about this work.

This work shows that using good object-oriented properties of the SEP model, we easily integrated a new model of computation. Then, it is possible to simulate heterogeneous systems composed of computational components, control-oriented components and continuous components. So studying complex systems, designers can use already proved components written with the esterel language and then focus on other components.

For a future work, an interesting point could be the investigation of properties we can deduce about other components from esterel components. In particular, we can study how relative order calculated by the esterel system can be used to deduce properties about other components.

We planned to use our method with an ASIC composed of a RISC and a DSP processor. It should be interesting to study how we can use the esterel language for the description of some of the contained components. By example, it may be possible to describe the processor controllers using a synchronous language. Then, we would just have to focus on computational units.

6. References

[1] *"Combining Special Purpose and General Purpose Languages in Real-Time Programming"*
C. André, A. Ressouche, J-M. Tanzi. Laboratoire I3S UNSA/CNRS. CMA, Ecole des Mines.

[2] *" Mixing Java and C in embedded systems"*
M.Bunnel. RTS'98, p.284-291, Jan 1998.

[3] *" Object-Oriented Extensions to CHDL : The LaMI Proposal "*
J. Benzakki, B. Djaffri. IFIP 1997. Chapman & Hall. p. 334-347.

[4] *" The esterel V5 langage Primer "*
G. Berry. Internal report 'Ecole des Mines' & INRIA/CMA. Avril 1997. http://www.cma.fr

[5] *" An Implementation of Constructive Synchronous Programs in POLIS "*
G.Berry, E.M.Sentovich. http://www.inria.fr/meije/esterel.

[6] *"Intégration de Modules Synchrones dans la Programmation par Objets"*
PhD thesis, Supélec/Université Paris-sud, Centre d'Orsay, Dec. 1993.

[7] *"Machines d'exécution pour langages synchrones"*
PhD thesis, Université de Nice-Sophia Antipolis, Nov. 1998.

[8] *" Un modèle fondé sur les assertions pour le génie logiciel et les bases de données "*
P.Collet. Thèse de doctorat, Université de Nice-Sophia Antipolis. Dec 1997.

[9] *" Conception d'un exécutif temps-réel pour esterel "*
D. Gaffé. Rapport de stage de DEA, Université de Nice-Sophia Antipolis. Septembre 1991.

[10] *" Hardware Modelling and Simulation using an Object-oriented Method "*
F. Mallet, F. Boéri, J-F. Duboc. Proceedings of the European Simulation Multiconference (SCS), Juin 98. ISBN 1-56555-148-6 p.166-168.

[11] *" Hardware Architecture Modelling using an Object-oriented Method "*
F. Mallet, F. Boéri, J-F. Duboc. Proceedings of the Euromicro'98. IEEE Vol I, p.147-153. ISSN 1089-6503.

[12] *" Inheritance Concept for Signals in Object-Oriented Extensions to VHDL "*
G. Schumacher, W. Nebel. Proceedings of the EURO-DAC'95 with EURO-CHDL'95.

[13] *"Object-Oriented Modelling and Design"*
J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen. Prentice Hall Inc. 13-630054-5, 1991.

[14] *"UML Notation Guide, Semantics and Summary"*
http://www.rational.com, Rational Software Corporation, 1997.