MODULAR FINITE STATE MACHINES IMPLEMENTED AS EVENT-CONDITION-ACTION SYSTEMS

Emanuel T. Almeida [1], **Jonathan E. Luntz** and **Dawn M. Tilbury** [2]

The University of Michigan
Department of Mechanical Engineering
Ann Arbor, MI 48109-2125
{almeidae,jluntz,tilbury}@umich.edu

Abstract: This paper presents a method to design a logic controller as an Event-Condition-Action (ECA) system, where the Modular Finite State Machine (MFSM) framework has been used to build the model for the controller. The resulting model, called an ECA MFSM, is in essence a MFSM model with a special structure that enables the processing of logic to follow the ECA paradigm. The ECA rule based method has a solid theoretical root and has been the paradigm followed to design active database systems. *Copyright© 2005 IFAC*

Keywords: Rule-based systems, Discrete-event systems, Database systems, Logical control, Finite state machines

1. INTRODUCTION AND MOTIVATION

The Modular Finite State Machine (MFSM) theory (Endsley and Tilbury, 2004*a*; Endsley and Tilbury, 2004*b*) has been developed for the design, verification and implementation of large scale logic programs, and to this end it has been successful. Control reconfigurability was also a major objective but has not achieved the same good results. This paper looks at logic architectures for MFSMs that enhance their reconfigurability, by turning the model into an Event-Condition-Action (ECA) system, which has been the underlying mechanism of a remarkably reconfigurable system, the active database.

Active database systems are designed to store large volumes of data and allow users or applications to manipulate the data in a controlled manner (Widom and Ceri, 1996). They are centered around the ECA paradigm that specifies the desired behavior for the database. In essence, when an event occurs a condition is evaluated (by a querying mechanism) and the database takes corresponding action (Zaniolo *et al.*, 1997). Active databases fall into the general category of *reactive systems* (Harel and Pnueli, 1989) and have an input/output behavior similar to a logic controller. If a major requirement for the logic control is reconfigurability, then it would be interesting if it shared the same underlying ECA mechanism as the active database. This mechanism is the way in which the logic controller internally handles and processes the logic. This paper shows how a MFSM logic controller can be designed so that its reactive behavior is specified by ECA rules, where the resultant model is termed ECA MFSM.

[1] Emanuel Almeida would like to acknowledge support from the Fundação para a Ciência e Tecnologia and the Fundo Social Europeu under the III Quadro Comunitário de Apoio

[2] This research was supported in part by the NSF under grant numbers EEC95-92125 and CMS98-76039

2. BACKGROUND

2.1 *Modular Finite State Machines*

MFSMs are a type of Discrete Event System (DES). They react to and generate events, and are an extension of FSMs, tailored to logic control applications, with added modularity and strong verification capabilities. As an example of a MFSM system, a simple plant consisting of a push button, a robot and a light is presented in Figure 1. The behavior of the system is that when the button is depressed the light will turn on and the robot will pick and place a part. Releasing the button at any time or depressing the button while the robot is busy will not have any effect. The light will turn off when the robot finishes its job.

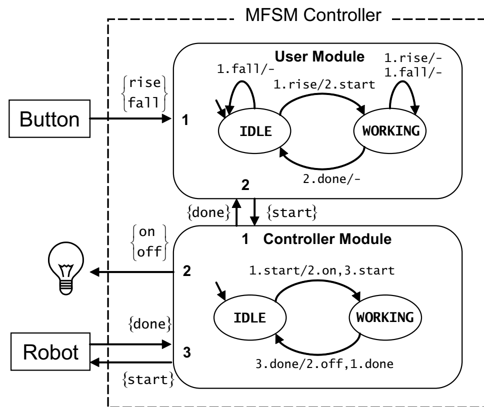

Fig. 1. Example of a MFSM controller

MFSM systems also fit into the broad category of reactive systems. External events (triggers) produce transitions in the MFSM model; these transitions can cause state changes in modules and generate responses back to the environment, as the effect of the trigger cascades through the MFSM model. It is assumed that this internal processing occurs instantaneously (or at least the time taken is negligible), such that no other external trigger can occur during this phase. This behavior is implementable in a scan based environment such as a Programmable Logic Controller (PLC).

An advantage of the MFSM framework is that it allows for controllers such as the one in Figure 1 to be modularly verified (Endsley and Tilbury, 2004*b*). However, previous work has revealed two weaknesses of the method. The major weakness is the complexity of the MFSM models, as can be seen for example in (Almeida and Tilbury, 2004) where the model for a cell controller developed for a manufacturing cell with only two machines and a robot can quickly become extremely large and difficult to understand. The second observed weakness is that the MFSM systems built up to now are bounded in their reconfigurability, meaning that the models can be easily modified only in control functions pre-conceived by the designer. Both of the observed weaknesses raise serious questions on how effective are the modular architectures proposed until now.

2.2 *Active Databases and ECA Rules*

Active database systems monitor events and trigger actions as a result of this detection. This behavior is specified in the form of ECA rules. A rule specifies that on the observation of a certain *event*, if a corresponding *condition* is satisfied, then an *action* is taken. An event can be a database operation, an external incoming event or a timed or untimed temporal event. The condition can simply be evaluated as true or false, whereas the action can be an action toward the environment or internal to the database and can be the trigger of a new ECA rule, causing a chain of rules to fire. ECA rules have been used in many domains ranging from business workflow managers (Bae *et al.*, 2004), manufacturing control (Chaudhry *et al.*, 1998) or web applications (Papamarkos *et al.*, 2003). The advantages of ECA rules lie primarily on the fact that they allow behavior to be specified and managed on a rule base (rather than being encoded in diverse applications), improving modularity, maintainability and thus reconfigurability. They have a generic and high level syntax easily understood and amenable to analysis, which makes them a natural candidate to implement reactive functionality.

3. EVENT-CONDITION-ACTION MFSMS

3.1 *General Overview*

Event-Condition-Action Modular Finite State Machines (ECA MFSMs) are MFSM systems with a special architecture of the modules. This particular arrangement makes the internal processing of the logic follow the ECA paradigm. The top part of Figure 2 shows a general representation of a MFSM system with input and output events. Below it is the ECA MFSM representation of the same system; the only change arises in the internal arrangement and design of the modules. In this new architecture there is a central module called **Main** and a finite number of **Peripheral** modules. Each one of these Peripheral Modules is connected to Main and prohibited from communicating with the environment or with other peripheral modules.

The ECA MFSM model interacts with the environment through a set of input events Σ_{in} and a set of output events Σ_{out}. Both event sets correspond to events that are exchanged with the plant, therefore form the set of *external* events $\Sigma_{ext} = \Sigma_{in} \cup \Sigma_{out}$. Given this general setup, the

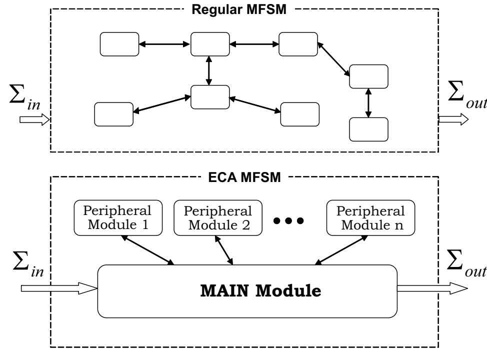

Fig. 2. A regular MFSM system with input and output events, and its ECA version

following sections formalize the various details of ECA MFSMs.

3.2 ECA MFSM Building Blocks

The various building blocks of ECA MFSMs consist of particular events, queries, states and modules, defined in this section. The event set of an ECA MFSM model is defined as the union of two sets: the set of *external* events Σ_{ext} (already defined) and the set of internal events Σ_{int}, which corresponds to events that are exchanged in between modules only. There are three subsets of internal events: the set of *query* events Σ_q, the set of *query-response* events Σ_r [illegible] updating events Σ_u.

Definition 1. An event belongs to Σ_q when it is a trigger to a Peripheral module and a response from Main and does **not** change the state of the Peripheral module; an event belongs to Σ_r when it is a response from a Peripheral module and a trigger to Main that is generated by a query event; an event belongs to Σ_u when it is a trigger to a Peripheral module and a response from Main that **may** change the state of the Peripheral module.

Definition 2. A Modular Query **Q** is performed between Main and a Peripheral module, through which connection the Peripheral module will have *one* trigger $\mathtt{c} \in \Sigma_q$ and n responses $\{\mathtt{d_1}, \ldots, \mathtt{d_n}\}$ $\in \Sigma_r$. It consists of Main sending $\mathtt{c}$ and the Peripheral module responding with one element from $\{\mathtt{d_1}, \ldots, \mathtt{d_n}\}$, resulting in one element from the set $\{\mathtt{cd_1}, \ldots, \mathtt{cd_n}\}$

There are two kinds of reachable states possible in a module, *stable* and *transient* states.

Definition 3. A state x is *stable* when the module can be at x at the beginning of a scan cycle; if the module can **never** be at x then x is *transient.*

In words, a stable state is a true system state, whereas a transient state is a temporary state that the module is in only in between scans, that is during the processing of the logic.

Definition 4. Main is a module where: 1)The initial state is the only stable state; 2)There is one connection to every Peripheral module. Through these connections, triggers to Main are query-response events and responses are query and updating events; 3)There can be one or more connections to the environment. Through these connections, triggers to Main are input events and responses are output events.

Definition 5. Peripheral modules have the following restrictions: 1)There is only one connection to Main per Peripheral module; 2)The triggers to Peripheral modules are query and updating events; the responses are query-response events; 3)Each time a Peripheral module receives: a query event it will issue one response to Main; an updating event it will not issue any response.

3.3 ECA MFSMs Behavior

This section explains how the *internal* events will come together with the *external* events to form *control transactions*, where these transactions are a are

Definition 7. A transaction graph has one stable state (IDLE) and multiple transient states, forming a tree-like structure as shown in Figure 3b. From any state there can be a transition to IDLE, thus originating cyclic behavior.

Definition 8. A control transaction $\mathbf{T}$ is a concatenation of ECA rules. It is composed of an event that triggers the control transaction $\mathtt{e} \in \Sigma_{in}$, of condition evaluation response / query pairs $(\mathtt{dc})_i$, $\forall i = 1, \ldots n-1$, of the updating actions to be taken $\mathtt{f}_i \in \Sigma_u$, $\forall i = 1, \ldots m$ and of the output actions to be taken $\mathtt{g}_i \in \Sigma_{out}$, $\forall i = 1, \ldots p$, where $(n, m, p) \in \mathbb{N}$, such that $\mathbf{T} = \{\mathtt{ec}(\mathtt{dc})_1(\mathtt{dc})_2 \ldots (\mathtt{dc})_{n-1}\mathtt{df}_1\mathtt{f}_2 \ldots$
$\ldots \mathtt{f}_m\mathtt{g}_1\mathtt{g}_2 \ldots \mathtt{g}_p\}$

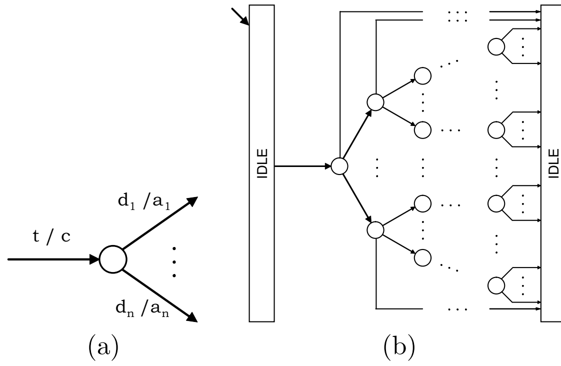

Fig. 3. a) Branching representation of a rule; b) For every $\mathtt{e} \in \Sigma_{in}$ there is a transaction graph representation with this general structure

There will be a transaction graph for every $\mathtt{e} \in \Sigma_{in}$; for convenience a rectangle is used to describe the IDLE state, and two instances of IDLE are shown, the first to show where control transactions start from and the second to show when a transaction terminates (finishes). The set of transactions of a system will be all possible paths for every $\mathtt{e} \in \Sigma_{in}$ that start from IDLE and end in the termination instance of IDLE. There exists one transient state in the transaction graph for every modular query that needs to be executed.

Definition 8 shows a concatenation of ECA rules until there is a response $\mathtt{d_i}$ that only causes updating and output events. As a whole $\mathbf{T}$ can be decomposed into a trigger component event $\mathtt{e}$, a condition evaluation component $\mathtt{c}(\mathtt{dc})_1(\mathtt{dc})_2 \ldots (\mathtt{dc})_{n-1}\mathtt{d}$ and an action component $\mathtt{f}_1\mathtt{f}_2 \ldots \mathtt{f}_m\mathtt{g}_1\mathtt{g}_2 \ldots \mathtt{g}_p$.

Figure 4 shows an example of a transaction graph. The triggering event $\mathtt{e}$ starts a query that can result in $\{\mathtt{c}_1\mathtt{d}_1, \mathtt{c}_1\mathtt{d}_2\}$; $\mathtt{c}_1\mathtt{d}_1$ will generate $\mathtt{f}_1$ and transaction termination; $\mathtt{c}_1\mathtt{d}_2$ will start a new query that can result in $\{\mathtt{c}_2\mathtt{d}_3, \mathtt{c}_2\mathtt{d}_4\}$. In this case $\mathtt{c}_2\mathtt{d}_3$ will generate $\mathtt{f}_2$ and $\mathtt{c}_2\mathtt{d}_4$ will generate $\mathtt{f}_3$ and $\mathtt{g}_1$ - in either case the transactions terminate. The same analysis is done in Figure 4 by explicitly

ECA Rules	Event	Condition	Action
Rule #1	e	c_1d_1	f_1
		c_1d_2	t'
Rule #2	t'	c_2d_3	f_2
		c_2d_4	f_3, g_1

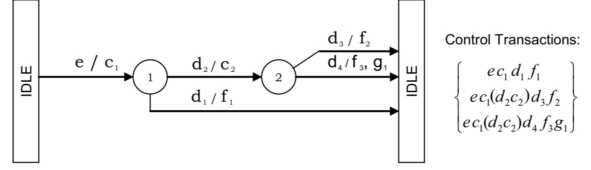

Fig. 4. An example of a transaction graph with the ECA rules explicitly written in a table and the 3 possible control transactions (or paths) outlined

writing the ECA rules for this transaction graph as a table (note that rule 2 is concatenated to one of the outcomes of rule 1, which means that rule 2 can *only* be triggered following rule 1, and that an event $\mathtt{t}'$ not represented in the transaction graph is used as a link between the action of rule 1 and the trigger of rule 2).

3.4 ECA MFSM System

Definition 9. An ECA MFSM system is composed of one Main module with exactly one ECA rule for every event $\mathtt{e} \in \Sigma_{in}$ and at least one ECA rule for every event $\mathtt{d} \in \Sigma_r$, and n Peripheral modules where $n \in \mathbb{N}$.

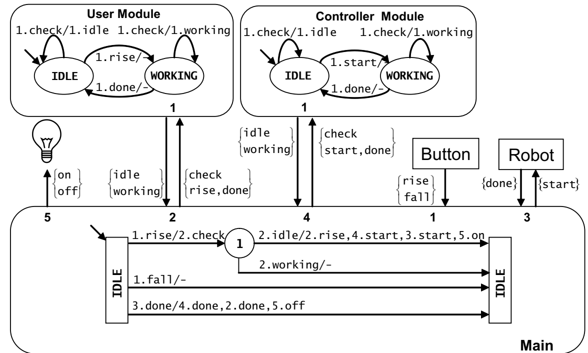

Fig. 5. The ECA MFSM version of the controller of Figure 1

Consider again the example of Section 2.1. A regular MFSM controller for this system is shown in Figure 1, and an ECA MFSM controller for the same system is shown in Figure 5, where Main has 4 control transactions.

3.5 Separation of state from logic rules

Consider the two MFSM controllers shown in Figure 1 and Figure 5, which are controllers for the same system that will produce the same input/output behavior (the parallel composition

of both systems is shown in Figure 6), but which have a clearly different architecture.

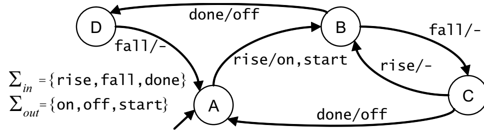

Fig. 6. Parallel composition of each of the controllers of Figure 1 and Figure 5

The MFSM in Figure 1 has 2 modules that are not designed independently; their logical behavior is based upon the protocol of events they will exchange. Therefore, **User** has to 'know' the **Controller** logic (and vice-versa).

The MFSM in Figure 5 has 2 Peripheral modules that perform the same state tracking as the modules in Figure 1 but are completely independent of each other, meaning they have no knowledge at all what the other module does (or even if it exists). Their joint behavior is tied by the control transactions in Main. The Peripheral modules perform the function of tracking state values whereas Main contains the ECA rules that query these modules in order to make logical decisions. There is a separation of functionality as the Peripheral modules become state keepers and Main becomes the decision making module.

4. COMPARISON OF DESIGN METHODS

We again turn to the example of Section 2.1, and add a Pause Button to it (the rise and fall events from this button are abstracted as the events `pause` and `continue`; it is assumed these events will occur alternately like the rise and fall events of a button). The occurrence of a single `pause` event would cause the light to remain on after the robot finishes its job after which further requests are ignored. The light in this case indicates that the robot is effectively still busy since it is in a paused state. `Continue` will bring the system back to normal mode.

In order to alter the controller in Figure 1 to handle the new functionality, the most direct way would be to alter the logic in **User** and **Controller** as shown in Figure 7a (grey states are new states added to the controller; italicized transitions are new transitions added). This involves changing pre-existing logic in the modules. Moreover it is necessary to understand how the logic of **User** affects the logic of **Controller** in order to make the changes, which implies understanding the system's global behavior.

Adding the pausing functionality to the ECA MFSM shown in Figure 5 can be achieved by adding a new Peripheral module **Pause** and altering existing control transactions and creating new ones (to handle the new events `pause` and `continue`), as shown in Figure 7b.

This is done by adding ECA rules without altering pre-existing rules (other than changing the triggering event of some rules), and without altering the logic in the pre-existing modules. Transactions are simply elongated by more queries, which are the reason for the new transient states 2 through 5, which reflect the length of the rule and not new stable states.

5. IMPLEMENTATION EXAMPLE

The Reconfigurable Factory Test Bed (RFT) (Moyne *et al.*, 2004) is a test bed at the University of Michigan used for research in networks and controls. There is a central controller called the System Level Controller (SLC), built as an ECA MFSM, that must manage hardware components for manufacturing operations, and software components for virtual factory control and diagnostics. Figure 8 shows the general layout of this controller as well as its connections with hardware and software components. The boxed region of Figure 8 contains the actual modules of the SLC, that consist of Main and 19 Peripheral modules, where Main contains approximately 80 transactions.

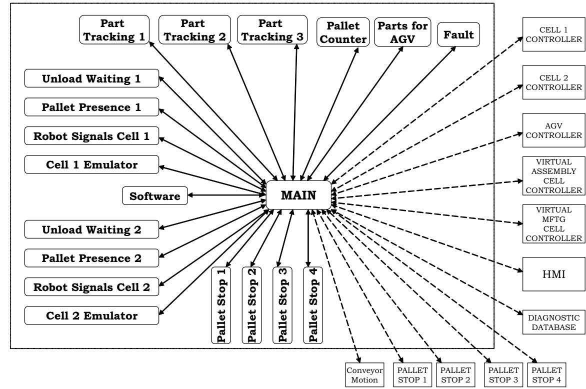

Fig. 8. ECA MFSM RFT Controller

This SLC was verified by placing protocols in the communication between the SLC and the various controllers it communicates with, and the state space found was of over 250,000 states. The reconfigurability of the SLC was tested when a faulting functionality was introduced into the controls; the SLC should shut down a manufacturing cell whenever the current read from a milling machine exited a certain diagnostic threshold, and parts should then be routed to a virtual manufacturing cell. The creation of a Fault module with only 2 states and the introduction of 4 new ECA rules (in a similar way as in Section 4) handled a potentially complex problem, and the SLC was then verified again for correctness.

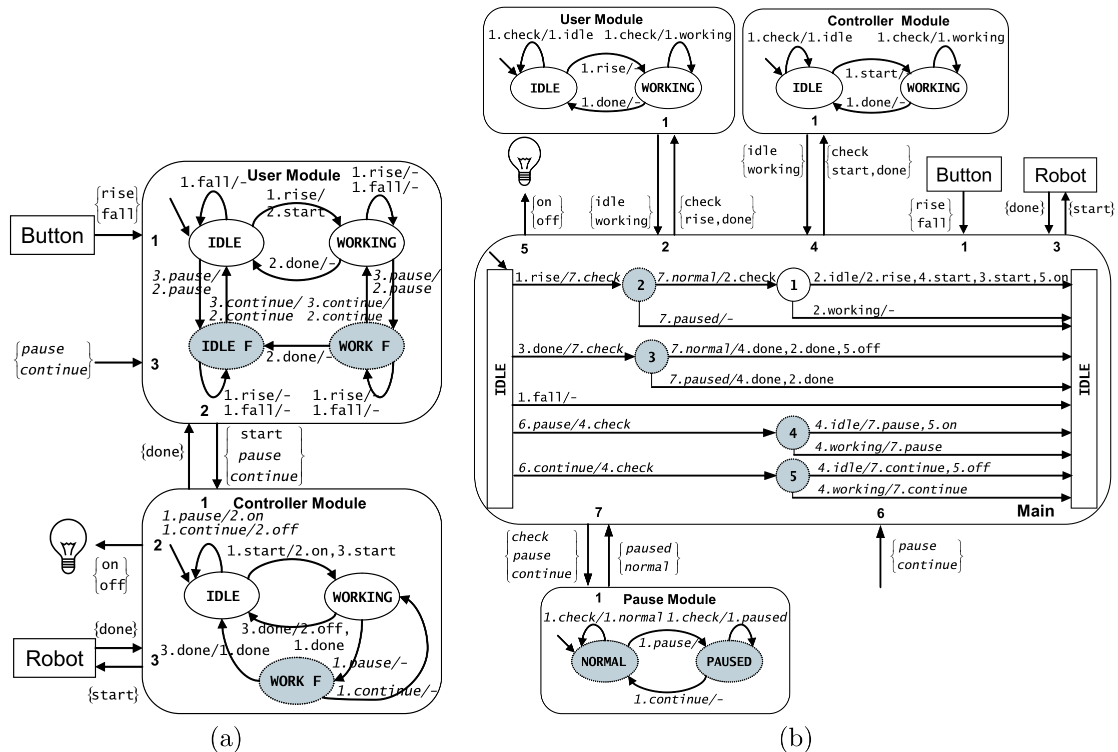

Fig. 7. a) Pausing functionality introduced by composing it into the modules; b) Controller of Figure 5 modified to have a pausing functionality

6. CONCLUSIONS

This paper has presented an approach to design MFSM controllers as ECA systems. ECA rules have been widely used in such domains as active databases, business workflow managers and web applications. They provide an excellent way to design and implement reactive systems, such as logic controllers. Reconfigurability has been one of the main drivers of this approach as modularity and maintainability are greatly enhanced when the logic functionality is managed within a single rule base rather than being encoded in diverse applications. This approach has also presented a clear modular architecture that can be used to design reconfigurable MFSM controllers. Building on the verification capabilities of MFSMs it is hoped that strong results can be obtained about the verification of certain classes of ECA systems.

REFERENCES

Almeida, E. and D. M. Tilbury (2004). Automatic code generation for reconfigurable cell-based manufacturing systems. In: *Proceedings of the IFAC Workshop on Discrete Event Systems*. pp. 31–36.

Bae, J., H. Bae, S.-H. Kang and Y. Kim (2004). Automatic control of workflow processes using ECA rules. *IEEE Transactions on Knowledge and Data Engineering* **16**(6), 1010–1023.

Chaudhry, N., J. Moyne and E. A. Rundensteiner (1998). Active controller: Utilizing active databases for implementing multi-step control of semiconductor manufacturing. *IEEE Transactions on Components, Packaging and Manufacturing Technology Part C: Manufacturing* **21**(3), 217–224.

Endsley, E. W. and D. M. Tilbury (2004*a*). Modular finite state machines for logic control. In: *Proceedings of the IFAC Workshop on Discrete Event Systems*. pp. 403–408.

Endsley, E. W. and D. M. Tilbury (2004*b*). Modular verification of modular finite state machines. In: *Proceedings of the IEEE Conference on Decision and Control*. pp. 972–979.

Harel, D. and A. Pnueli (1989). On the development of reactive systems. *NATO ASI Series F: Computer and Systems Sciences* pp. 477–498.

Moyne, J., J. Korsakas and D. M. Tilbury (2004). Reconfigurable factory testbed (RFT): A distributed testbed for reconfigurable manufacturing systems. In: *Proceedings of the Japan-USA Symposium on Flexible Automation*.

Papamarkos, G., A. Poulovassilis and P. T. Wood (2003). Event-condition-action rule languages for the semantic web. In: *Proceedings of the Workshop on Semantic Web and Databases*.

Widom, J. and S. Ceri (1996). *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan-Kaufmann.

Zaniolo, C., S. Ceri, C. Falutsos, R. T. Snodgrass, V. S. Subrahmanian and R. Zicari (1997). *Advanced Database Systems*. Morgan-Kaufmann.