Realization and Real-time Properties of Reconfiguration and Transient Management Methods*

[1] Tamás Kovácsházy, Gábor Péceli, Gyula Simon
{khazy,peceli,simon}@mit.bme.hu

[2] Gabor Karsai
gabor.karsai@vanderbilt.edu

— October 21, 2002 —

Technical Report

BMEMIT2002-TRAN02 (v1.0)

[1] Budapest University of Technology and Economics, Budapest, Hungary
Department of Measurement and Information Systems
http://www.mit.bme.hu/

[2] Vanderbilt University, Nashville, Tennessee
Institute for Software Integrated Systems,
http://www.isis.vanderbilt.edu/

* Funded, in part, by DARPA's Software-Enabled Control Program under AFRL contract F33615-99-C-3611.

Table of contents

List of Figures

1 Introduction

System reconfigurations are of major concern in embedded control applications, where the effects of internal or external changes may ask for drastic modifications within the architecture and the operation of the controllers. In dynamic systems, these kinds of modifications are followed by transient phenomena resulting in possibly unacceptable side effects. For this very reason, reconfiguration methods should include also some measures concerning transient behavior. These measures are referred to as transient management. In this report some possible realization schemes of certain reconfiguration methods including transient management are investigated. The primary aim of this work is to give aspects of performance characterization of system reconfigurations, and to provide a conceptual framework to support their development and design.

The problem to be solved here is how to move a real-time system from its actual configuration to a new one with predictably low magnitude and short time transient response. There are several activities, which might influence significantly this reconfiguration procedure. To provide a proper design of this procedure we must be aware of the complete system as much as possible both in design-time and also in run-time. The complete system in our case includes both the plant and the controller, or equivalently the complete physical environment and the embedded computational system. Changes, i.e., intentional or unintentional reconfigurations might occur in any part of the overall system. Therefore, during operation an ongoing real-time identification might be required to detect these changes, and decision-making mechanisms are to be invoked, which might propose some ongoing controller design, and initiate reconfiguration actions. While investigating realization schemes, all these activities are strongly interrelated, and should be executed in a timely manner. For this very reason, throughout this report the term reconfiguration, if not indicated otherwise, refers to all these interrelated issues.

In this report, however, the main emphasis is laid on the implementation of transient management methods. An attempt is made to characterize how the reconfiguration of discrete time, linear dynamic systems (e.g., filters, PID controllers, etc.) can be solved within a predefined reconfigurable system architecture. General issues of timing, concurrency and complexity are analyzed, the results of which might help in solving resource management problems both in design-time and run-time.

In section 2 the major activities related to reconfiguration of dynamic systems are described. Section 3 introduces the system architecture and a reference system, which is used throughout this report as an example to illustrate and compare the properties of the alternative solutions. The one-step reconfiguration method is investigated in section 4. Section 5 describes the multiple-step reconfiguration-with-state-preservation method. A relatively smooth transition can be achieved if we operate both the old and the new configurations, and systematically blend the outputs. This blending method is examined in section 6. In section 7, the real-time performance of the some initial state computation methods is described. Anti-transient signal generation closes the list of investigated transient management method in section 8. Finally, the conclusions are drawn in Section 9.

2 Activities related to reconfiguration

There are two major types of reconfiguration processes having somewhat different scenarios:

- Reconfiguration process due to an unintentional (unexpected, unpredicted) change within the system. The cause of the unintentional reconfiguration can be an error or failure (within the controller or the plant), e.g., sudden degradation of sensors or actuators. In this case, first the change in the system has to be identified and then a decision is to be made how to reconfigure the controller, and how to reduce the reconfiguration transients. The majority of these steps are run-time actions asking for intensive on-line processing.
- Intentional (planned and possibly scheduled) reconfiguration process. The need for such a reconfiguration is known well in advance, and its architectural forms and parameters are mainly design-time issues. Typical examples are the regular, operational mode changes. The reconfiguration process is initialized typically by event signals coming from the higher layer sub-systems of the controller or from the user.

2.1 Unintentional change in the system

The following activities are to be performed following an unintentional change within the system:

1. Identification of change (or "diagnosis"),
2. Decision making,
3. Controller re-design,
4. Reconfiguration and run-time transient management.

The identification procedure takes some time, because the estimation of the changed system parameters will be possible only if the system behavior diverges from the nominal behavior perceptibly after the change.

Having the new system parameters, the controller decides how to modify itself to compensate the (intolerable) effects of the sudden change (decision making), and works out its new configuration (design). As the final step the controller reconfiguration and transient management are to be done (intervention, corrective action).

The detailed timing diagram of the activities is shown on Fig. 1. The activities depicted as gray are unavoidable in that epoch: the system has to perform these tasks. Typically the white segments are not utilized, except in certain practical cases, and where execution can be done in parallel with other activities. For example, in the majority of the cases the system identification should be stopped during reconfiguration, and also for a given time after reconfiguration, because the reconfiguration transients might mislead the system identification procedure, and generate continuous oscillation between modes. In other scenarios, it is possible to stop the identification just after the detection of the change within

the system, and thus possibly improve resource-utilization by scheduling other, important, reconfiguration-related tasks and activities.

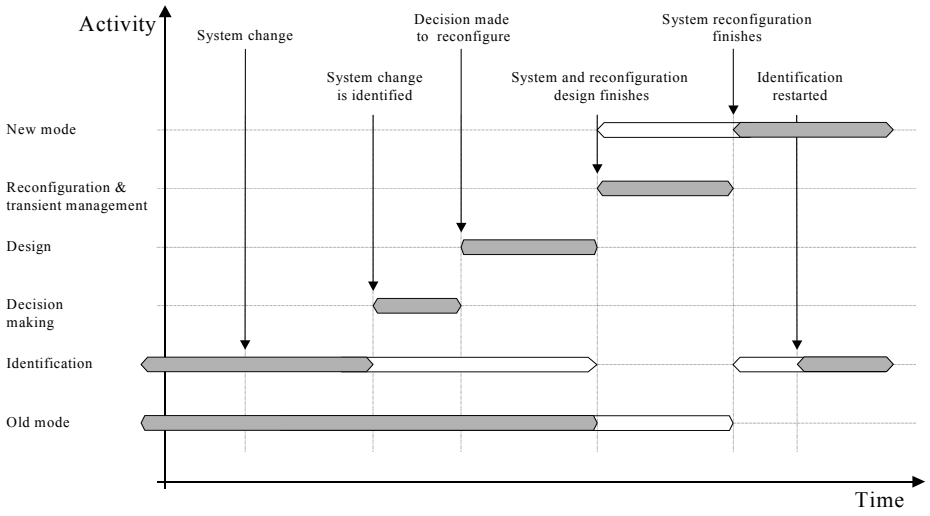

Fig. 1 Timing diagram of activities related to reconfiguration done after an unintentional change in the system

2.2 Intentional reconfiguration

The scenario of the intentional reconfigurations is somewhat simpler, only the design of the new controller and the reconfiguration including transient management are to be performed. The event, which initiates reconfiguration, is associated with the parameters of the new plant and the control objectives, and based on these data a new configuration is selected and parameterized, and finally the reconfiguration, together with the transient management, is performed. The selection of a new configuration is a design activity, however, in some practical cases, it is possible to pre-design the controller for certain intentional events, which makes it possible to omit the design phase resulting in lower resource utilization and reduced reconfiguration time overhead.

Even in the case of intentional reconfiguration, identification is present as an activity before and after the reconfiguration, because the system must be ready to do reconfigurations in case of any unintentional change. As previously mentioned, the identification should be stopped before the reconfiguration happens, and it can be restarted after the reconfiguration transient settles, otherwise the identification may identify the reconfiguration as an unintentional change, and start follow up reconfigurations putting the system into an unstable mode.

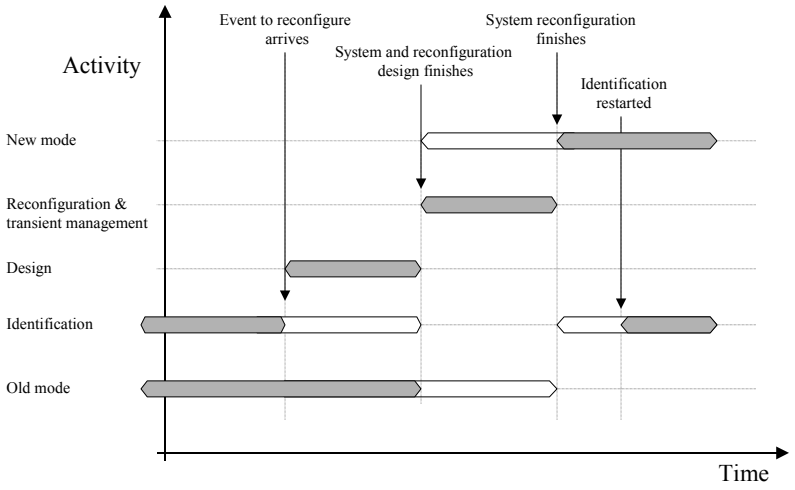

Fig. 2 Timing diagram of activities related to an intentional reconfiguration

2.3 Reconfiguration and transient management

In Fig. 1, and Fig. 2 reconfiguration and run-time transient management are not shown detached from each other, because, depending on the available information, the relative timing can be different for these two activities.

- If we are aware of the system to be changed by reconfiguration, and we know the time schedule of the complete scenario, we can apply *a priori transient management* methods, which, by perturbing/biasing the system before the changes, result in smaller "energy" transients [PK99].
- If we do not know the exact time of the reconfiguration in advance, only *a posteriori transient management* methods can be applied, i.e., such techniques, which directly utilize information available only at the time instant of reconfiguration.

Obviously the first approach can be combined with the second one. Understanding of the relative timing of the reconfiguration and run-time transient management is essential, because it shows clearly when the execution of the transient management algorithms will require additional resources.

Fortunately, the identification activity is typically stopped during the reconfiguration, so it is likely that resources in excess are available to execute the transient management and reconfiguration algorithm. On the other hand, the identification, transient management and reconfiguration algorithms are not necessarily executed on the same hardware in a distributed system; therefore further considerations might be essential.

3 System architecture

As shown before, there are various parallel and/or sequential activities to be done during the operation of a reconfigurable system. These activities are realized by separate components, which communicate with each other to achieve the proper operation. The relation of these components, and their interaction closely connected to the real-time properties of the algorithms used in the components; therefore, it is necessary to take into account the architectural differences, and possible realizations. Furthermore, the complexity of reconfigurable systems must be handled with proper and straightforward system partitioning, which can be derived from the distinct activities.

In addition, a reference system is defined in this section, which is used as a common application throughout the report with slight modifications to accommodate the differences of the transient management and reconfiguration methods.

3.1 Reconfigurable system architecture

The inherent complexity of reconfigurable systems can be dealt with by introducing a component-based, layered architecture. One possible partitioning is shown on Fig. 3, which is used in the FACT framework [FWWW]. In this report, the right side of Fig. 3 is in the center of interest, because all the transient management and reconfiguration related components/layers are there.

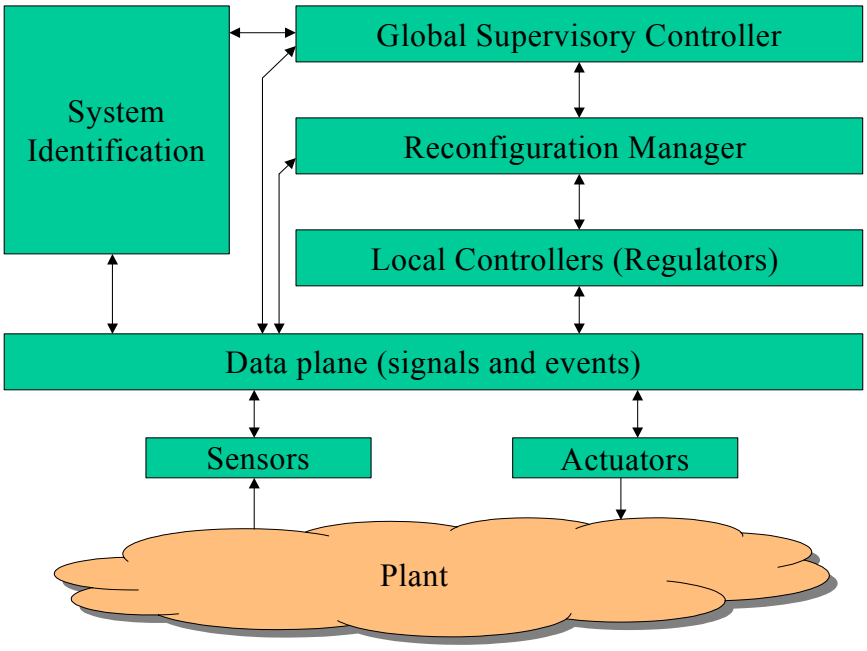

Fig. 3 Components of the reconfigurable system

The components in Fig. 3 are the following:

- Global supervisory controller (GSC),
- Reconfiguration manager (RM),
- Local controllers (LC)
- Sensors and actuators,
- System identification,
- Data plane.

The report concentrates on the GSC, RM and LC components. The GSC component does not directly influence the real-time performance of the transient management and reconfiguration, because its main role:

- To capture global modes of the system,
- To produce global, high level control signals and events (primarily used by the RM and LC components, and not by the plant directly).

Therefore, it performs primarily the high-level decision making related activities during reconfiguration. The GSC has not only the knowledge of the system-level modes, but also the current system-level control objectives. Obviously the GSC performs its activities in real-time, but the real-time performance of the GSC is not investigated in this report. The GSC is envisioned as a complex hierarchical state-chart [HAR87].

The GSC can make higher-layer-design related decisions too, but the detailed controller design is mostly the task of the RM and LC components. The RM component:

- Acts as an intermediary between the GSC and LC:
    - By mapping GSC modes to LC modes (not necessarily a one to one mapping),
- Does global transient management,
    - By incorporating transient management related information and temporary modes, and synchronizing the transient management related activities of LCs,
    - By doing transient related decision-making and design.

The LC components encapsulate the regulators and all other regulator specific functionalities, such as low-level regulator design; from low-level, implementation specific parameter computation from higher-level specification to more complex non real-time design; regulator level transient management, and regulator reconfiguration. Therefore, the LC components:

- Are connected to signals and events on the data plane,
- Operate on and produce signal and event inputs and outputs based on local criterion by incorporating regulators (the global criterion are incorporated by the configuration),
- Consist of the local logic (local supervisor) to do local (and implementation specific) reconfiguration and local transient management,

- Incorporate design procedures.

In distributed systems requiring distributed control, the LCs may be real controller boxes near the plant, connected to certain sub-systems of the plant; for example, one or more LCs may be assigned to a control surface of an airplane. While the RM and GSC components are higher-level components running typically on higher speed central computers, but they can be distributed as well.

The data plane acts as a virtual, real-time, configurable "wiring closet" to distribute signals and events to the interested components coming from the signal sources. This data plane can be realized by a low-level communication network with such higher layer protocols on each node (implementing the LC, RM, GSC, etc.), which provide guaranteed real-time performance.

In the above-mentioned layered architecture the GSC component issues reconfiguration commands to the RM component to start reconfigurations. The GSC can wait for the RM component to complete reconfiguration, in which case the reconfiguration is called *synchronous*, or it may enter into the next state without waiting, which is the case *asynchronous* reconfiguration. These communication primitives are captured within the statechart of the GSC as new pseudo states. The synchronous reconfiguration indicated with a hexagonal form in the GSC statechart, while the asynchronous reconfiguration is captured with a triangle form, see the GSC component in Fig. 4 for examples.

The *synchronous* reconfiguration makes possible to synchronize the GSC and the RM/LC layers during the complete process of reconfiguration, and therefore complex, long reconfigurations and context based error handling is possible. The *asynchronous* reconfiguration lets the GSC to advance after issuing the reconfiguration command; therefore, it limits the capability of the system from the point of view synchronization, while its implementation is simpler and easier.

3.2 Reference system

A simple, conceptual system (controller and plant) is constructed to show the operation of the GSC, RM and LCs, and to be used as reference system during the investigation of transient management and reconfiguration methods. The task of this system is to regulate a plant with two sensor and actuator pairs using a reconfigurable scheme. The regulation of the plant can be done using PID controllers attached to the sensor and actuator pairs, as seen in Fig. 4.

The plant has two global modes (S1 and S2), and the configuration related to these modes is known in design time, therefore the PID controllers and transient management can be totally specified in design-time, no run-time system design is required. There are dedicated reconfiguration manager sub-systems for both mode changes: one for going from mode S1 to S2 (synchronous), and one for going from mode S2 to S1 (asynchronous). The transient management methods applied within the components implementing the reconfiguration managers (RM12, RM21) are identical in this example, but real applications may ask for different transient management and the reconfiguration strategies. One reconfiguration is synchronous and the other one is asynchronous from the point of view of communication (between the GSC and RM) to show the differences resulted from this behavior, if there are any.

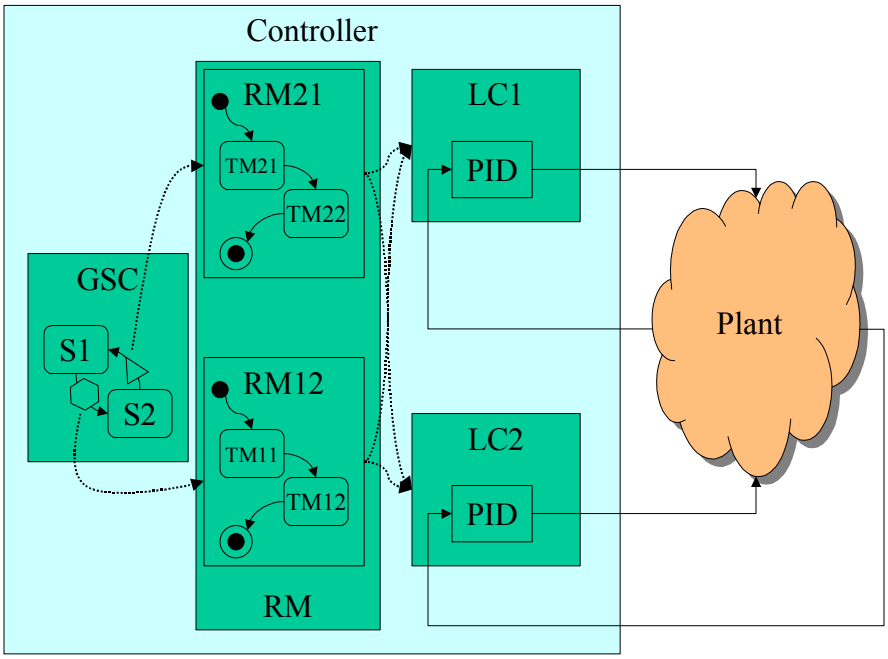

Fig. 4 Conceptual control system

3.3 Realization level of transient management

The transient management algorithms can be realized in the RM, or in the LC components, or in both of them.

- If the LCs are complex enough to handle transient management, then the main task of the RM is the consistent and synchronized mapping of the GSC modes to LC modes.
- If the LCs do not provide transient management, and they provide only a standard way to access the regulators, the RM has to handle also the transient management related activities.
- In certain applications, it may be necessary to do transient management both on the LC and the RM level. In these cases, the LCs deal with the low level, short term, local transient management, and the transient management on the RM level addresses primarily the global, longer-term requirements.

The decision to put the selected transient management technique into the LC or the RM level depends mainly on the application and the information available. Some basic considerations are listed in Table 1.

	RM level implementation	LC level implementation
Run-time information used	Globally available run-time information, such as large number of plant and LC inputs and outputs	Locally available run-time information, such as internal states, previous inputs and outputs
Synchronized execution of the reconfiguration	Synchronization of the reconfiguration of multiple LCs is possible	No trivial synchronization of the LCs is possible
Complexity	Complex algorithms are possible, because the resources used to perform identification are freely available in this phase	Depends on the available resources, but primarily lower complexity algorithms are allowed only
Requirements	Global, generic requirements	Local, regulator specific requirements
Generality	General algorithms may be formulated on the RM level, and may be used to reconfigure LCs providing standard reconfiguration methods	The reconfiguration techniques may be general, but for performance reasons it might be necessary to dedicate them to the actual regulator

Table 1 Considerations to implement a transient management technique on RM or LC level

4 One-step reconfiguration with state preservation

The one-step reconfiguration with state preservation is a reconfiguration method, which does not include run-time transient management. The simplest form of this method does not affect the structure of computational model of the system to be changed; only the parameters (e.g., filter coefficients) are modified (see Fig. 5). This approach implies that the run-time task of the RM and the LCs is only to change the coefficients of the regulator.

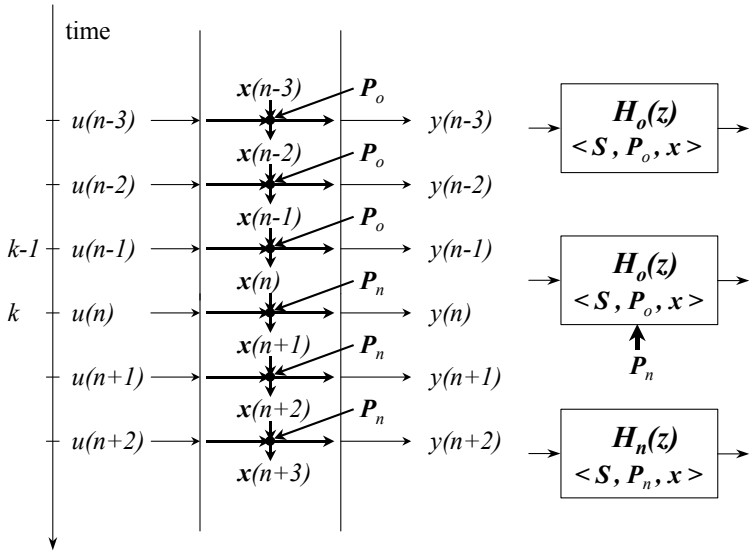

Fig. 5 One-step reconfiguration by changing the coefficient of the regulator during operation

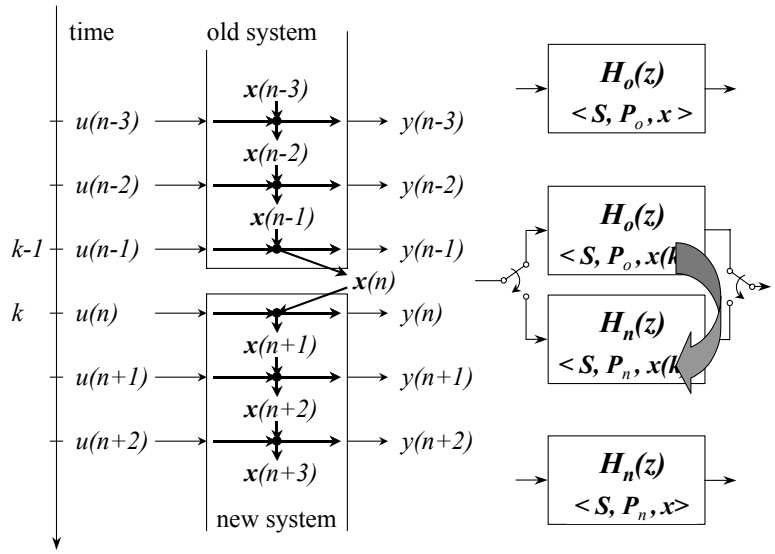

Fig. 6 One-step reconfiguration by copying the states and doing structural modifications

A more complicated version of this basic method involves structural changes (see Fig. 6), as well. The preservation of the state variable values is solved in such a way, that the state information of the old configuration is simply fed into the new one, without any further considerations. This step is followed by the activation of the new configuration. The initialization of the state variables is somewhat ad hoc, however, it is important to note, that in this case there are no associated run-time computing costs; the "new states" are computed by the old system.

4.1 LCs with configurations generated at design-time

Here it is assumed that no run-time design occurs, i.e., we can assume that the new configurations of the PIDs are available in the LCs, and the RM only orders the LCs to change configuration. The configurations are downloaded into the LCs during the system initialization.

This can be implemented by swapping pointers/references identifying different sets of coefficients, for example, in high-level computer languages such as C/C++ or JAVA. In addition, this reconfiguration method can be implemented in programmable logic devices too, by swapping register files or other memory blocks, which is an easy to implement and computationally efficient realization. Therefore, these realizations show very limited computational overhead compared to non-reconfigurable systems. Of course, the configuration data should be stored, which increases storage requirements in the regulators linearly with the number of configurations (generated at design time). In addition, to keep LC level consistency, the swapping of the coefficient in the LC should be done when the LC does not use the coefficients to compute its outputs. Furthermore, if multiple LCs are reconfigured, it is necessary to keep all LCs in a consistent configuration.

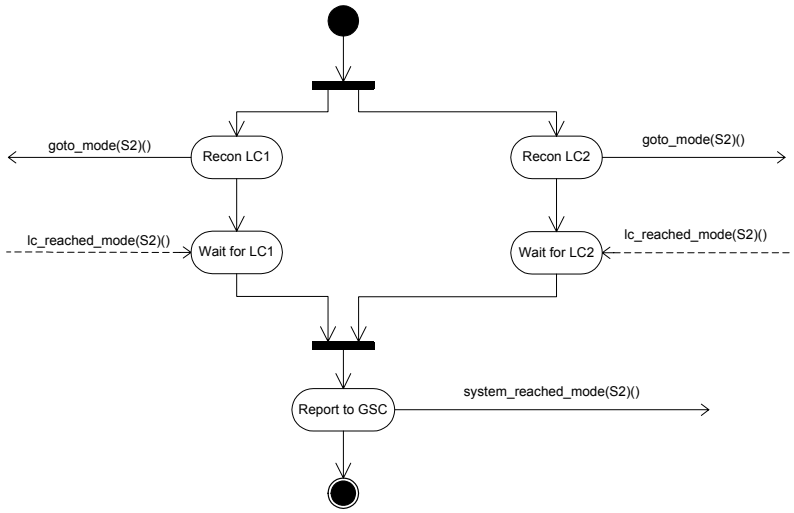

Fig. 7 Activity diagram of the RM for the one-step reconfiguration from mode 1 to 2, when the GSC does a synchronous reconfiguration

The internal activities performed in the RM are influenced by the communication scheme used on the GSC level:

1. The GSC may require synchronous reconfiguration, in which case, the RM should wait for the LCs to complete the reconfiguration. The GSC can leave the synchronous

reconfiguration mode when the RM sends a system_reached_mode message. See Fig. 7 for an activity diagram detailing the RM used to switch from mode 2 to 1.

2. If the GSC requests an asynchronous reconfiguration, the RM needs to send out only a reconfiguration request to the LCs. See Fig. 8 for the activity diagram.

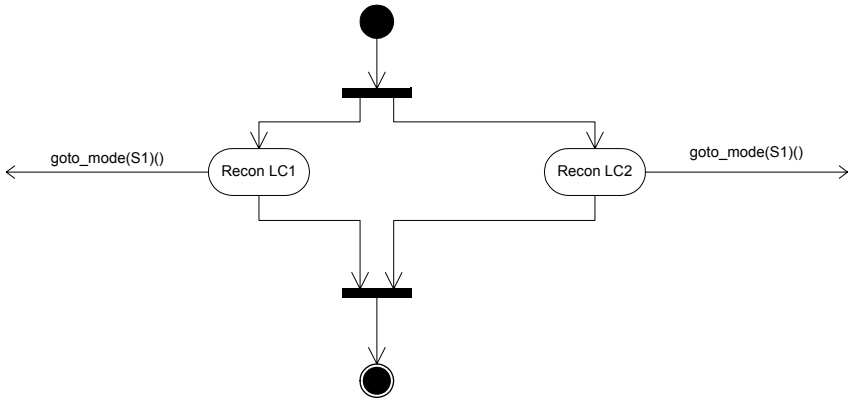

Fig. 8 Activity diagram of the RM for the one-step reconfiguration from mode 2 to 1, when the GSC does a asynchronous reconfiguration

4.2 LCs with configurations generated at run-time

The activities detailed in Fig. 7 and Fig. 8 are valid only if the PIDs are designed at design-time, and stored in the LCs, in other words, if run-time design is not needed. If run-time design cannot be avoided, a more complex sequence of activities has to be implemented. See Fig. 9 for a revised activity diagram for the run-time design case.

The reconfiguration, and copying of the coefficients can be considered negligible from the point of view of resource needs and real-time properties compared to the design activity; therefore, the real-time properties of this method are primarily determined by the design phase, which is investigated in details in section 5 for discrete time, linear, dynamical systems.

First, the new configurations are designed and downloaded into the LCs. Some activities may be executed in parallel. In general, we assume RM level design, and the LCs may do some simple mapping of the downloaded LC parameters to the coefficients of their internal algorithm. When all the configurations are designed and downloaded, it is possible to do such a reconfiguration, which has the same real-time properties as the previous case (no design).

If we store the configuration on the RM level, or we allow regulator design, the consistency of the individual configuration must be maintained. In essence, the modification of an individual configuration-change must be treated as an atomic operation assuring that the regulator cannot operate with an intermediate configuration, in which a part of the coefficients are appropriate for the old, and other part of the coefficients are appropriate for the new configuration. If the RM to LC communication does not guarantee the conditions of atomic coefficients changes, the LC should provide it.

Another prerequisite comes from the synchronization of the reconfiguration of multiple LCs. The LCs should be in a consistent mode, and this must be guaranteed in most of the cases. By sending out goto_mode(S1) and goto_mode(S2) in parallel, this consistency is not

necessarily kept, one of the LCs can go into S1 sooner than the other, and in some cases it can lead to instability and/or other problems within the plant. Here we assume that the goto_mode messages and the operations done in the LCs as a result of the goto_mode message are executed instantaneously for all LCs, i.e., the PIDs are reconfigured at the same time.

The activity diagrams do not capture erroneous operation, when exceptions arise; they assume that all activities are completed properly and within the prescribed time. Error handling would make the diagrams more complicated; and in addition, it would distract the attention from the investigated timely, functional operation. Furthermore, the detail level of error handling depends highly on the actual application, and the real-time aspects of corrective actions cannot be described in general.

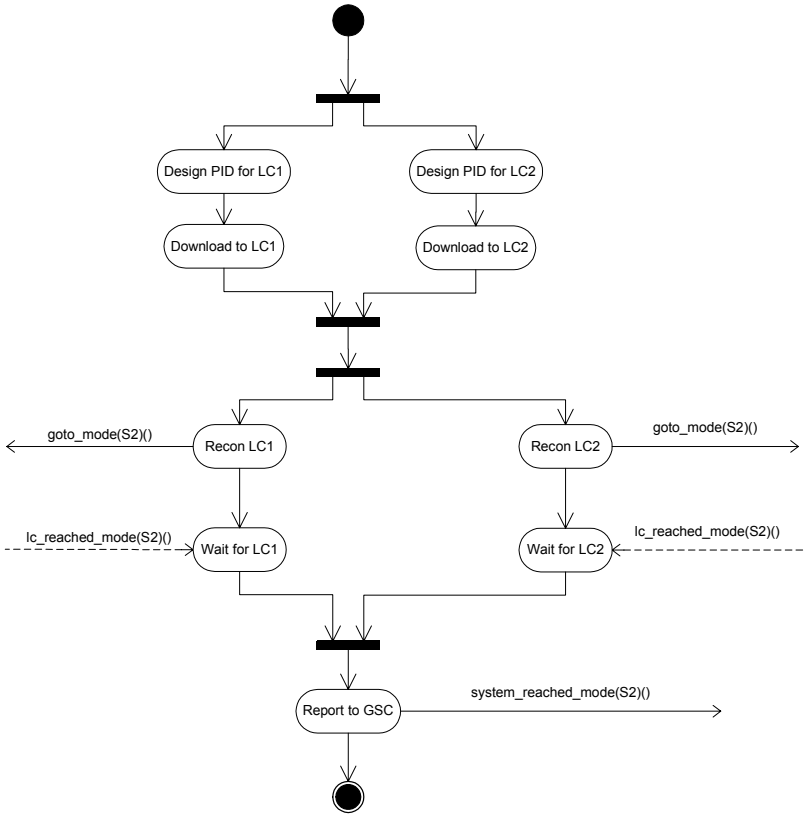

Fig. 9 Activity diagram of the RM for the one-step reconfiguration from mode 1 to 2, when the GSC does a synchronous reconfiguration and run-time PID design

5 Multiple-step reconfiguration with state preservation

The multiple-step reconfiguration uses temporary configurations between the old and the new configuration. The system is reconfigured using the one-step reconfiguration with state preservation to change these configurations, i.e., the multiple-step reconfiguration method is a series of one-step reconfigurations.

The transients and computational complexity of the multiple-step reconfiguration depend primarily on the method used to compute the temporary configuration. A possible method to compute the intermediate configurations is the linear interpolation of the coefficients of the regulators. This method is used here to demonstrate the activities done during multiple-step reconfigurations in the RM and LCs. More complex methods may linearly interpolate between higher lever design parameters, such as the cut-off frequency of filters, and design the filter according to these temporary parameters. It is even possible to search for the optimal temporary configurations at run-time, however for this type of algorithms the timely execution cannot be guaranteed.

5.1 Complexity of regulator design

There are various types of algorithms for intermediate configuration design, but they can be classified into three main groups from the point of view of their real-time properties:

1. Coefficient interpolation schemes, which have $O(N)..O(N^2)$ computational complexity (where N is the order of the regulator) and minimal additional storage space requirements,
2. Higher-level design parameter interpolation and detailed regulator design based on the interpolated parameter, which are more complex than the simple coefficient interpolation schemes, but in the majority of the cases they do not involve activities with unpredictable execution time,
3. Optimal temporary configuration search algorithms, for which it is hard to guarantee strictly bounded execution time.

All these solutions can be used to pre-compute temporary configurations, which trades run-time computational complexity for storage requirements, if the old and the new configurations are known in advance.

The simplest coefficient interpolation scheme linearly interpolates the coefficients of the temporary regulators between coefficients the old and the new system. This solution has $O(N)$ computational complexity, and requires some additional storage. Unfortunately, in some cases this method causes the reconfigured system to loose stability temporarily [PK99]. Similarly, if we get relatively close to the stability margin, the transient properties of these simple coefficient interpolation schemes might become unacceptable. The stability can be maintained if we apply structurally passive filters [MR76, PK99], which automatically assure stability during linear interpolation of the coefficients. Some more complex interpolation schemes may use non-linear, or higher-order interpolation techniques, which require higher computational complexity and storage requirements.

The transients can be reduced much better if higher-level parameter interpolation is used, which guarantees more adequate temporary configurations. For example, the cut-off frequency of a low-pass filter can be interpolated linearly between the old and new cut-off frequency during reconfiguration, thus resulting in proper low-pass filters for all temporary configurations. However, as a next step for all temporary configurations, the coefficients of the system must be computed from the higher-level parameters. The computational complexity and storage requirements of this process are highly regulator dependent. As an example, we may consider the implementation of the above-mentioned low-pass filter using different, e.g., direct, lattice, resonator-based, or parallel filter structures [PMP96]. Let us assume that the transfer function of the low-pass filter is known because that is computed first from the interpolated cut-off frequency. The coefficients of the direct structure are known in this case (they are the same as the a_i and b_i coefficients of the transfer function); the coefficients of the parallel and lattice structure can be computed with a worst-case $O(N^2)$ complexity, while the computation time needed for of the resonator-based structure, which involves finding of roots of polynomials, might be unpredictable.

Concerning optimal temporary configuration search algorithms only some preliminary propositions based on [KP2001] are available yet; this problem is subject to intensive research.

5.2 LC level transient management

In this case, the LCs implement the complete multiple-step reconfiguration method. By reconfiguring them to the new configuration, they automatically start changing the configurations, and reach the new configuration in a predefined number of steps. The number of steps can be specified as a parameter of the goto_mode() message, or can be computed based on the "distance" of the configurations, or can be in design time.

The LC sends lc_reached_mode() message if the new configuration is reached. Therefore, in this case the activity diagram of the RM is identical to the one-step reconfiguration, as seen in Fig. 7, Fig. 8, and Fig. 9, because only the activities done in the LC are different.

The LC level activities are detailed in Fig. 10. The system is driven by the incoming input samples, because the transient management related activities are closely synchronized to the sampling process. In other words, the system configuration needs to be evaluated for all input samples. If in the actual step the system is not reconfigured, it computes only the regulator output and states. If the system is to be reconfigured, it will compute the regulator outputs and states, using the old temporary configurations, in parallel with the new temporary configurations, then it modifies the regulator configuration from the old one to the new one.

The complexity of the LC level system design should be kept as simple as possible, preferably in the range of $O(N)$, where N is the order of the regulator. For example, simple linear interpolation of the coefficients has this complexity; the required number of operations (multiplications and additions) on the coefficients is in the range of 2N to 10N, the actual number is defined by the used filter structure. The temporary configuration design depicted in Fig. 10 is executed for all new samples. If the temporary configuration design needs more resources to execute than available it may be split into smaller sub-phases, which can be executed one after the other resulting a new temporary configuration in each i^{th} iteration (after any i^{th} new input samples).

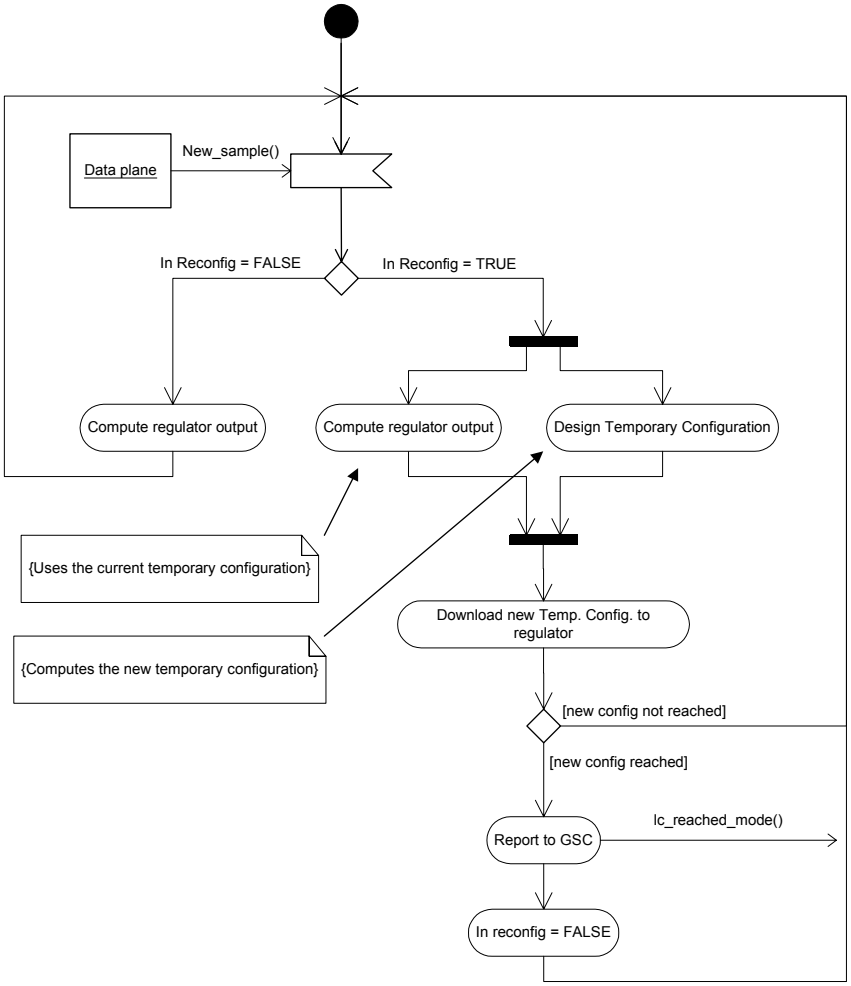

Fig. 10 LC level multiple-step reconfiguration activity diagram

5.3 RM level transient management

Multiple-step reconfiguration on the LC level does not allow complex synchronization schemes to be used to reconfigure multiple LCs. Furthermore, an LC level multiple-step reconfiguration method may work out intermediate configurations, which are not optimal in the global sense, unless the LCs have global information is provided for the LCs. The RM level approach allows the controller to take into account some global requirements and act in a synchronized way during reconfiguration. It may happen, that the LCs support only one-step reconfiguration with state preservation, in which case, the further reduction of transients asks for the use RM level multiple-step reconfiguration.

The activity diagram shown in Fig. 11 depicts this type of globally optimized design approach. As the first step "Design Global Temporary Configuration" activity is executed, and a global temporary configuration worked out. As a next step, the PIDs are designed according to the global temporary configuration, and the PIDs are downloaded to the LCs as new configurations. Then the LCs are reconfigured into the new temporary configuration. If synchronous reconfiguration is requested, the RM sends system_reached_mode() message

to the GSC as the final step. This sequence of activities is executed as long as the new configuration is not reached.

The design phase must be finished in time to allow timely reconfiguration; therefore, the intervals between the reconfigurations must be set according to the expected transient properties and the available resources to compute the new temporary configurations. On the RM level, it is possible to synchronize the reconfigurations to the incoming new samples, as it is done on the LC level, but less stringent synchronization is possible by guaranteeing reconfigurations periodically, e.g., using an RM level timer event, or other periodic event sources.

Because the multiple-step RM level reconfiguration is a multiple one-step reconfiguration, all remarks made there apply here, too. Therefore, the consistency of the individual configuration must be maintained and the LCs must be kept in a consistent mode. This is assured by executing the coefficient change command in an atomic way and instantaneously for all LCs.

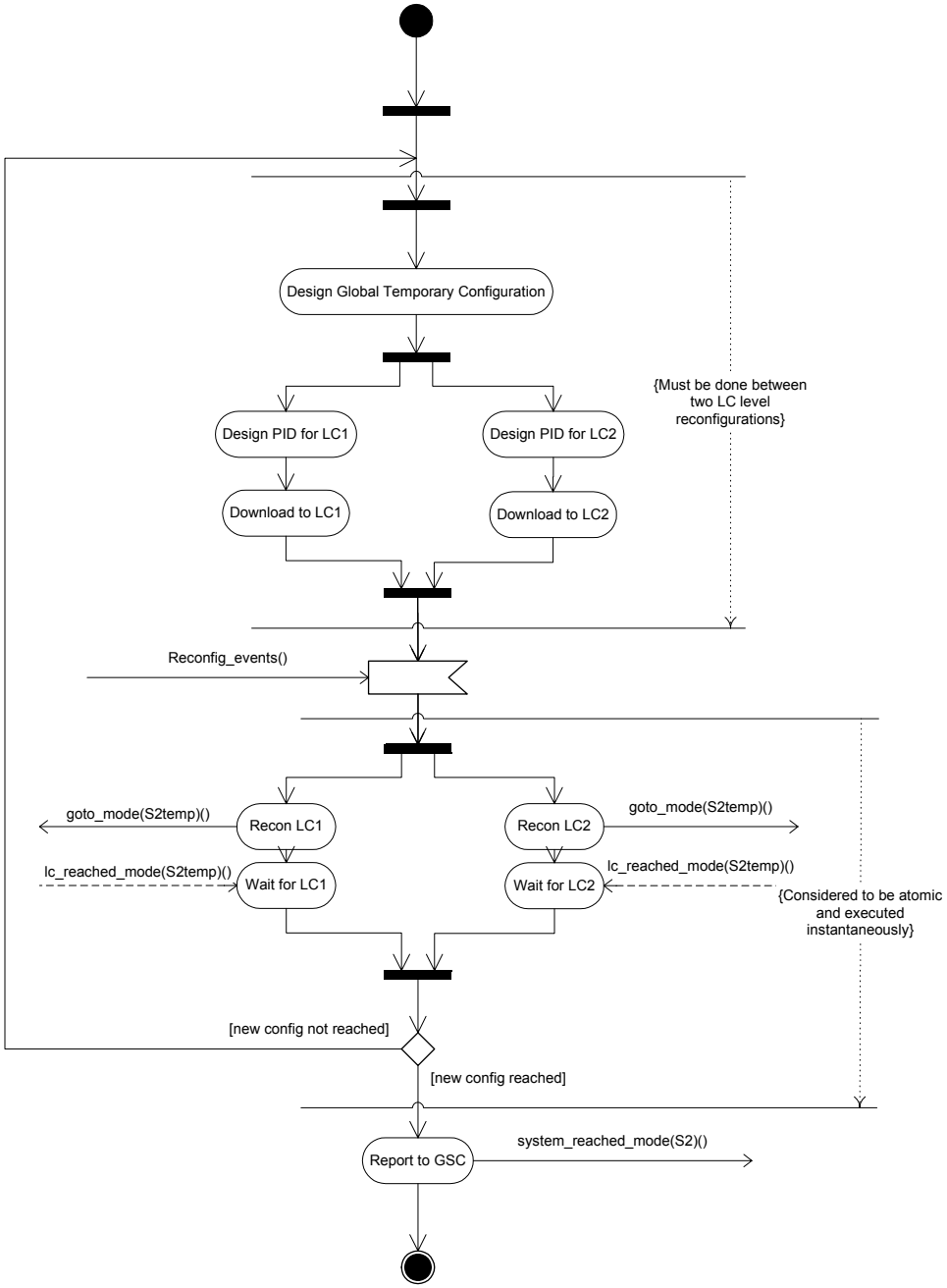

Fig. 11 Activity diagram of the RM for the multiple-step reconfiguration from mode 1 to 2, when the GSC does a synchronous reconfiguration and run-time temporary PID design

6 The method of blending

The method of blending changes the system structure during reconfiguration by inserting the new configuration in parallel with the old one, and smoothly replaces the old configuration with the new one by blending the outputs of the two systems.

6.1 LC level blending

The application of the blending method on the LC level needs a minor modification of the reference system. The LCs need to be modified to include the parallel PIDs and the blending algorithms at the outputs as shown in Fig. 12. The activity diagram of the RM is identical in this case to the RM activity diagrams of the one-step reconfiguration shown in Fig. 7 and Fig. 8. The RM has to realize the mapping between the GSC and the LCs.

The activities done on the LC level during blending are shown on Fig. 13. Blending needs to be synchronized to the sampling procedure to guarantee smooth transition from the old system to the new one. The duration (expressed by the number of samples) required for this transition, may be supplied by the RM as a parameter of reconfiguration.

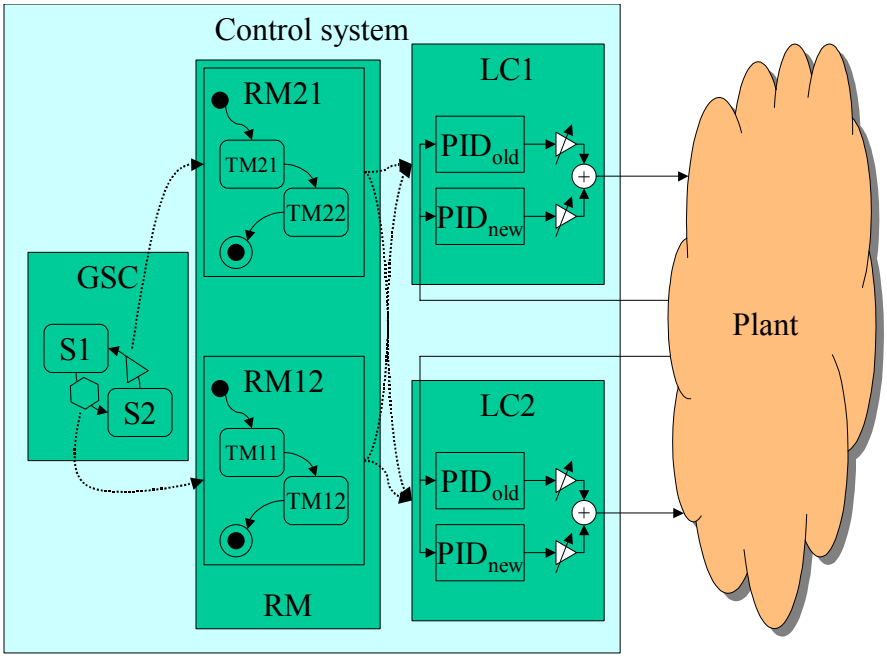

Fig. 12 Blending on the LC level in the reference system. Note that the LCs support blending by incorporating two parallel PIDs and the blending algorithm on the output.

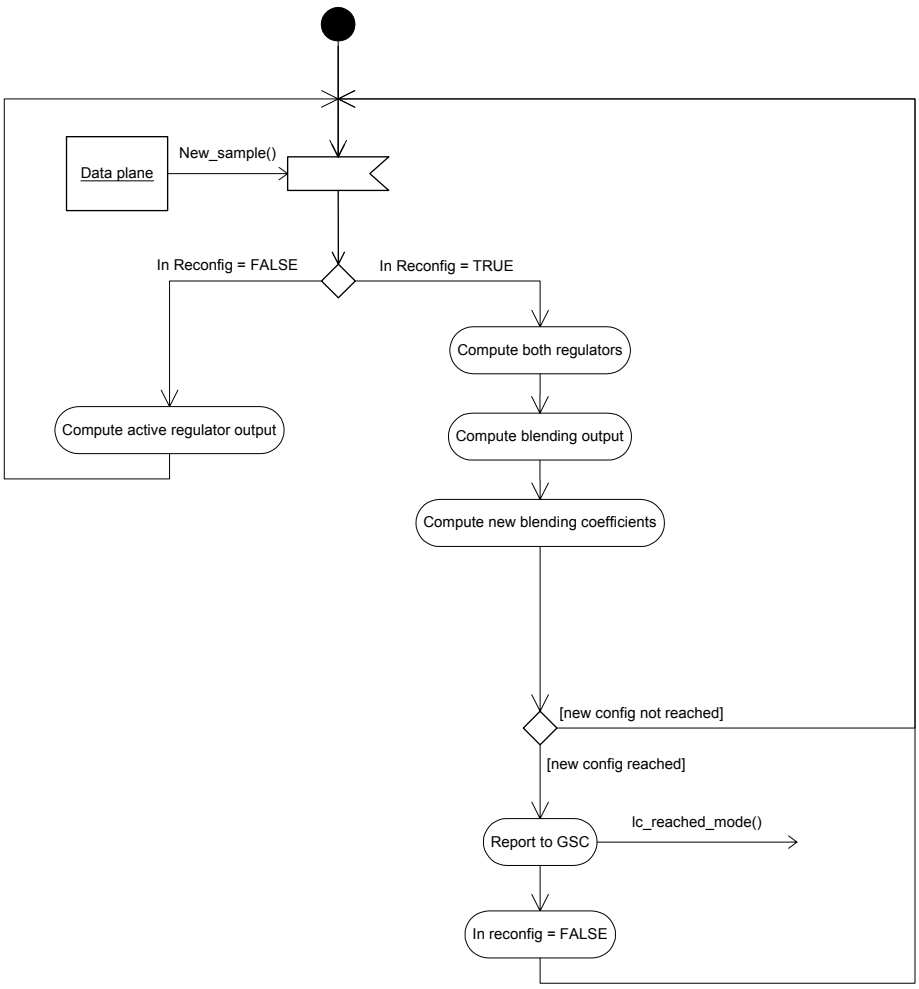

Fig. 13 Activity diagram of LC level blending

6.2 RM level blending

Blending on the RM level requires basic changes in the reference system, which is shown in Fig. 14. It is necessary to split the local controllers into smaller, more specialized local controllers, and reconfigure them in a synchronized manner. There are two possible realizations of blending on the RM level form the point of construction and destruction of LCs:

1. Design-time construction and destruction of LCs. All LCs are constructed at design-time, therefore all of them run parallel at run-time. This option is similar to the LC level blending; only the blender LCs must be reconfigured in a synchronized manner. The activities done in this case can be derived from the LC level blending, just by moving all activities to the RM level.
2. Run-time construction and destruction of LCs. Only the necessary LCs are executed, and therefore they must be constructed and destroyed run-time if required. See Fig. 15 for the RM level activity diagram.

Run-time construction of LCs include the following activities:

- Construction of the run-time components (LCs),
- Downloading the initial configuration of LCs into the LCs,
- Placing the new run-time components into ready-to-run mode.

A ready-to-run mode of the LCs can be described as a mode, in which the LC is ready-to-run, but not signaled, because it is not connected to the data plane.

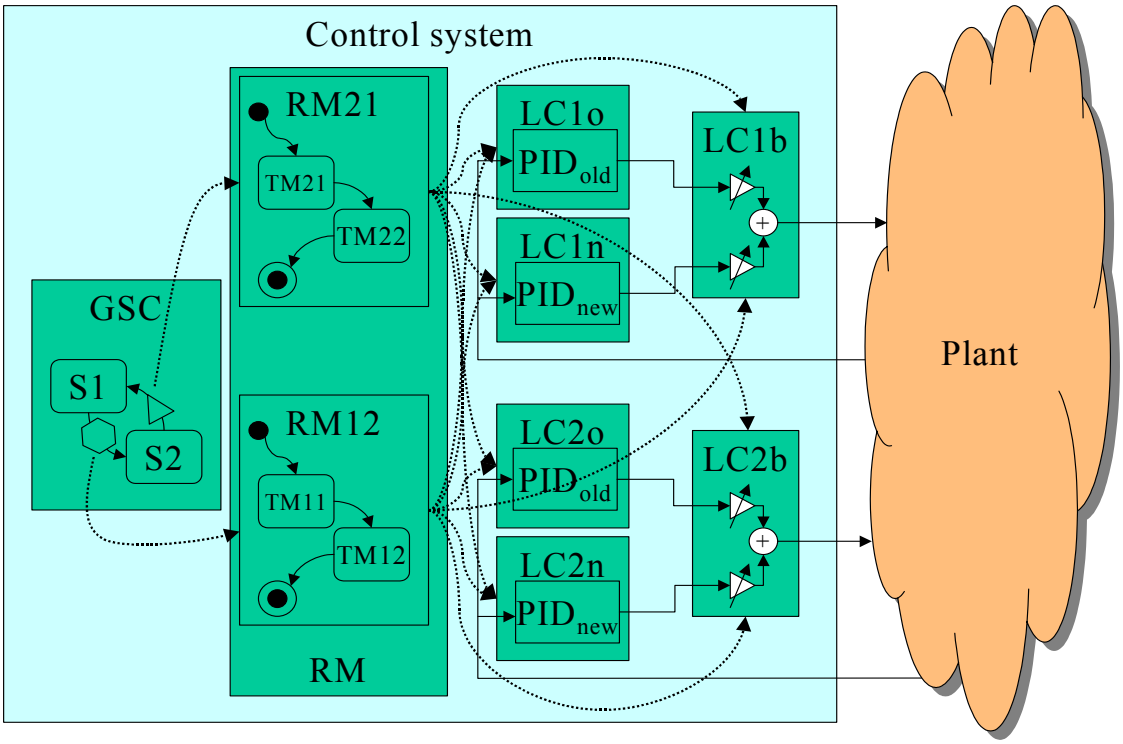

Fig. 14 Blending on the RM level in the reference system

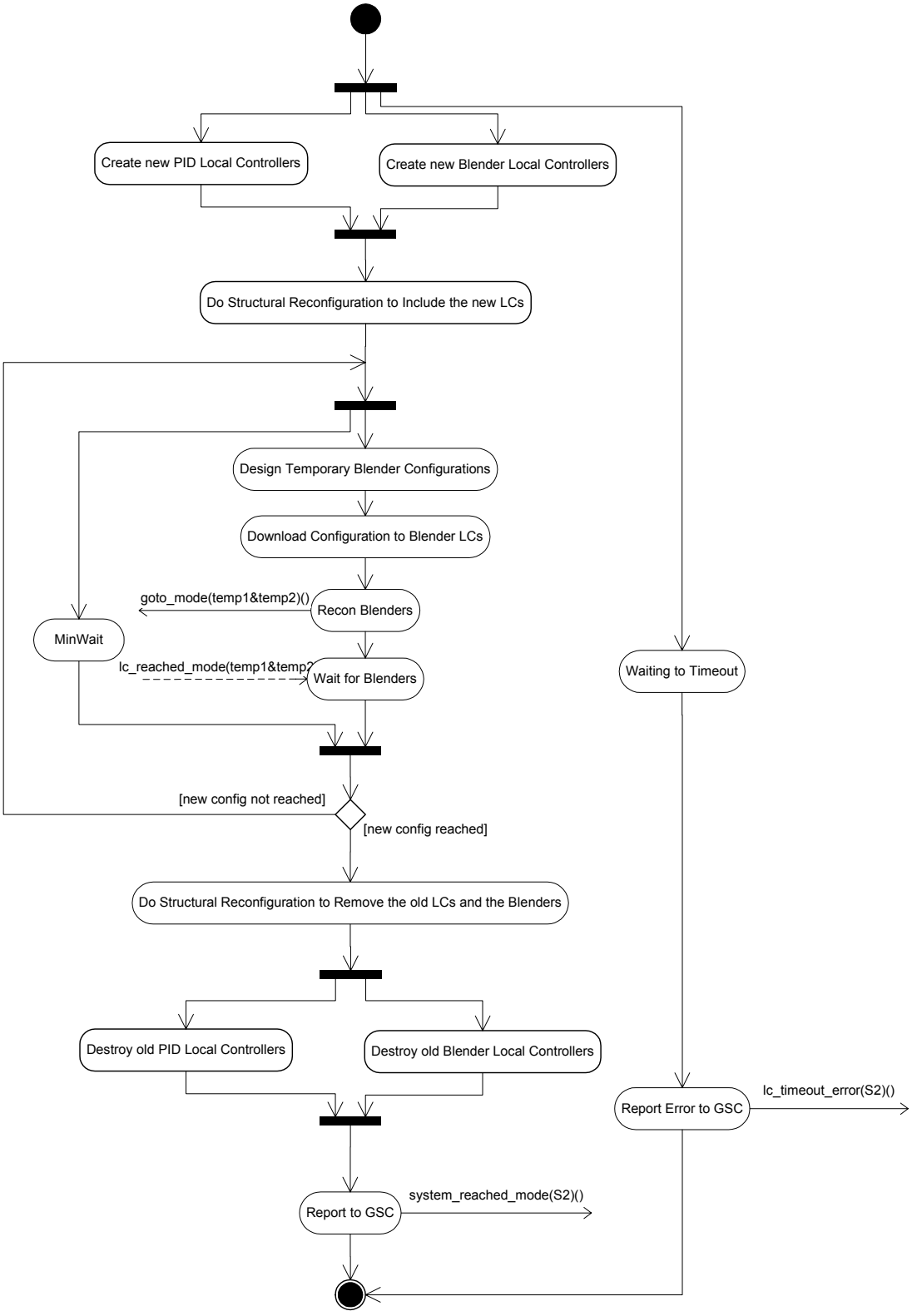

Fig. 15 Activity diagram of RM level blending done during a reconfiguration from mode 1 to 2

7 Computation of the initial states

Computation of the initial states (CIS) is a pure transient management method; it needs to be applied together with an appropriate reconfiguration method. It uses local knowledge, such inputs and internal states, and therefore it can be considered primarily for LC level implementations. In the majority of the cases the one-step reconfiguration method is used to change the controllers, as it is shown in Fig. 16.

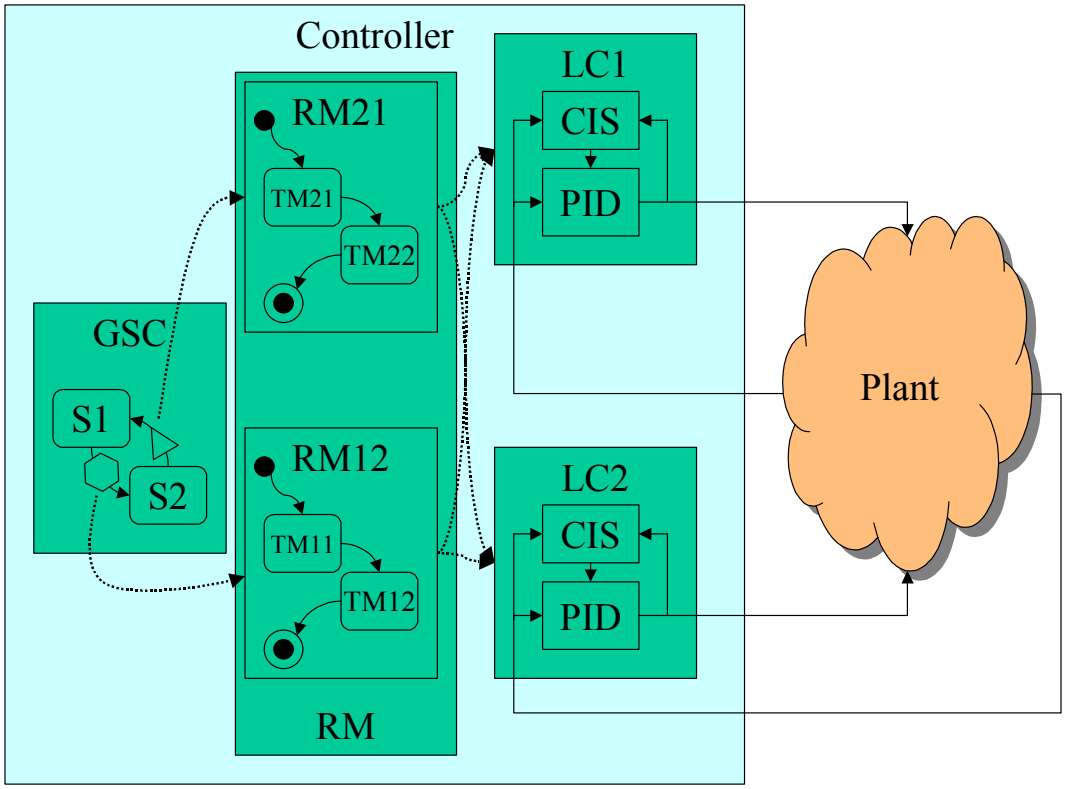

Fig. 16 Computing the initial states on the RM level in the conceptual control system

The initial internal states can be computed:

- at the reconfiguration of regulators instantaneously (Fig. 17),
- or before the reconfiguration of regulators using some kind of a filtering scheme (Fig. 18).

The operation depicted in Fig. 17, which assumes the computation of the initial states at the reconfiguration, at one time instant, between processing the two consecutive input samples coming in before and after the reconfiguration. The instantaneous computation has $O(N^2)..O(N^3)$ computational complexity because typically it requires to solve a set of linear equations [SPK02]. Here we must note that this computation of initial internal states must be

completed before starting computing the new outputs after reconfiguration, thus forming a real-time constraint for the system.

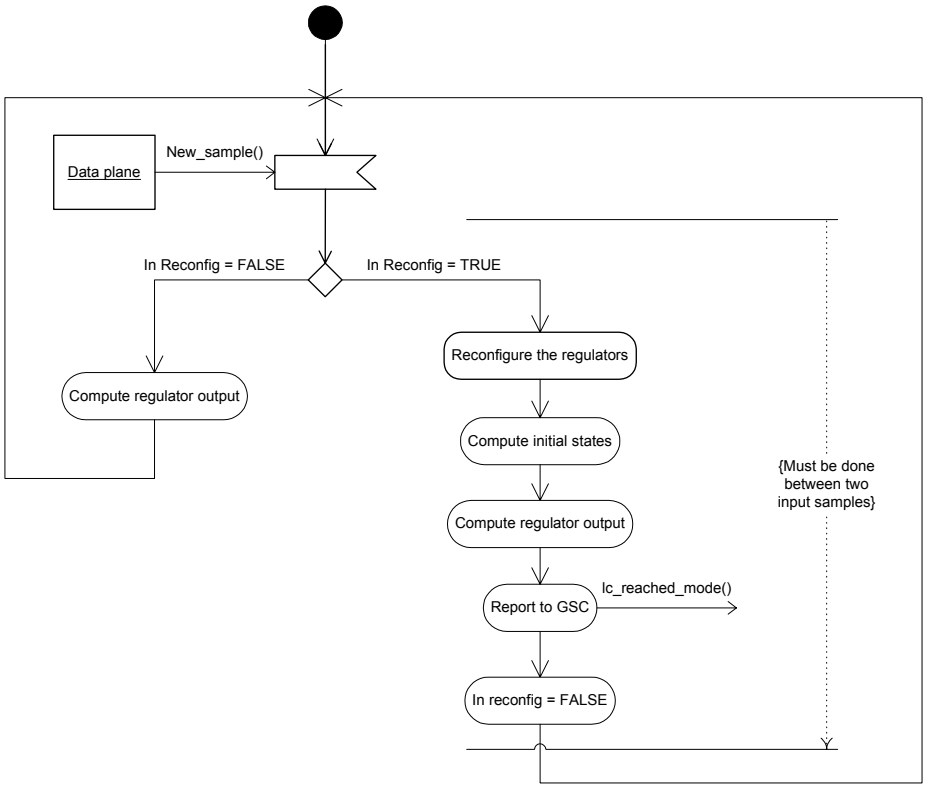

Fig. 17 Activity diagram of the LC level computation of the initial states at the reconfiguration

The filtering scheme introduced on Fig. 18 exploits the fact that sets of linear equations can be solved recursively, with $O(N)..O(N^2)$ computational complexity. On the other hand, this scheme delays the reconfiguration, because the system needs at least N iterations (input samples) to compute the proper initial states. The algorithmic details and underlying operation of the filtering scheme is also discussed in [SPK02].

Due to its high computational complexity, the instantaneous computation of the initial states requires large amount of free resources are available. It should be used in systems, which do not allow delaying the reconfiguration. The alternative filtering scheme is likely to used in all other cases with less stringent timing, or with resource limitations.

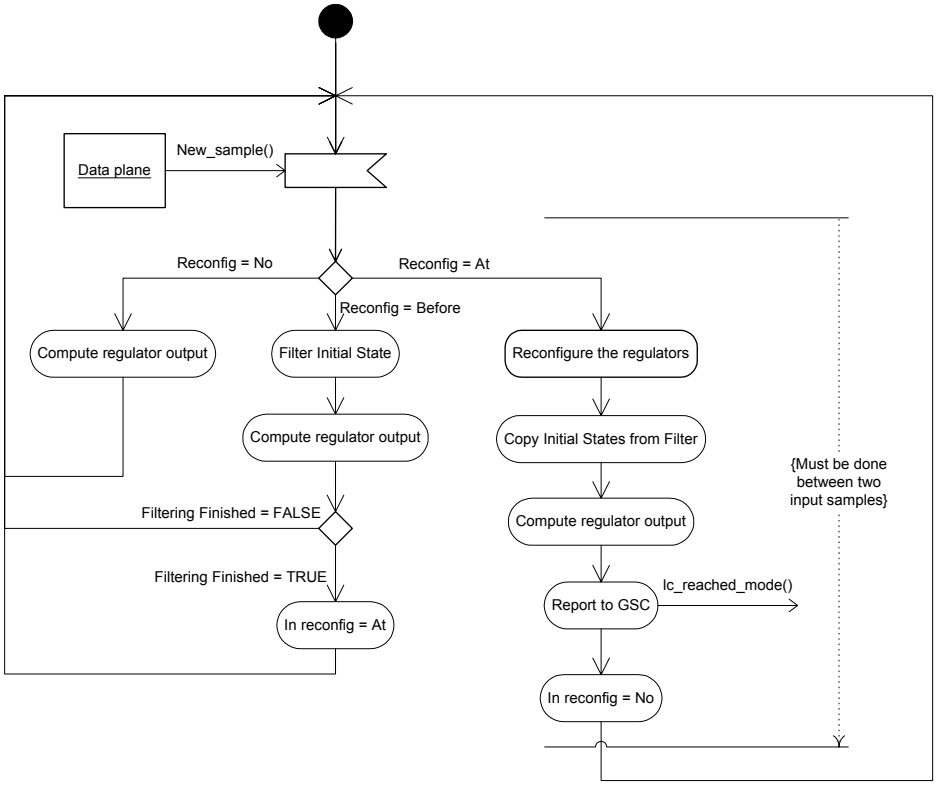

Fig. 18 Activity diagram of the LC level computation of the initial states by filtering before the reconfiguration

8 Anti-transient signal injection

Anti-transient signal injection is a pure transient management technique, and it is envisioned primarily as a method to reduce transients caused by system components within the plant to which we do not have direct and internal access[1]. Just because it is primarily for these types of systems, it is applicable on the RM level, where global information is available. The anti transient signal can be injected into the system:

- Before the system changes, i.e., the controllers or the plant reconfigured or changed (presumes an intentional or predictable change in the system),
- After the system changes (allows the identification of the change, such as an error),
- Both (before and after the reconfiguration, and also presumes intentional or predictable change).

Injection of the anti transient signal before the reconfiguration makes the system ready for the change. If it is combined with an anti transient signal generated after reconfiguration, further reduction can be achieved.

The anti transient signal generation to the reference system introduced in section 3.2 should be arranged as shown in Fig. 19. Anti transient signal injection requires the close synchronization of the plant (and its model), the controllers and, the anti transient signal injector (ATSI) component. In the majority of the cases the anti transient signal has to be generated based on the actual values of the state variables. The ATSI component adds the generated anti transient signal to the controller output, and those two forms together the input of the plant.

Two distinct activities are present during the application of the anti transient signal generation:

- First, the ATSI component must be designed and constructed (at time of starting the reconfiguration),
- Second, the ATSI component generates the anti transient signal.

If the reconfiguration is made intentionally, and both the old and new system models are known a priori, then the computation of the coefficients of the ATSI can be computed at design-time. In all the other cases the run-time computational complexity at the reconfiguration is high, and strongly depends on the used algorithms, but typically higher than $O(N^2)$.

The run-time computational complexity is $O(N^2)$, because the ATSI component does matrix multiplications [SPK02].

[1] The internal states, internal behavior, executed code or algorithms cannot be accessed, set or changed directly as in case of the typical digital implementations of controllers and other components.

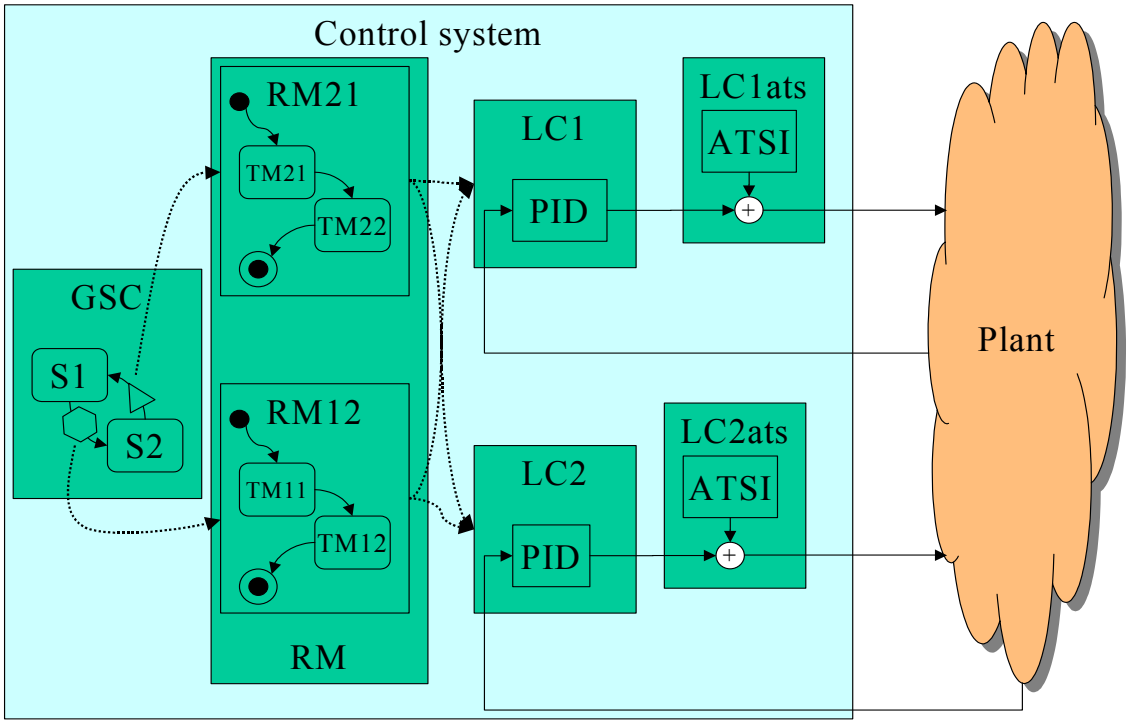

Fig. 19 The reference system modified for the application of anti-transient signal injection on the RM level

The activity diagram of the post system change anti transient signal injection is shown on Fig. 20 for systems in which the anti transient signal is generated by an autonomous dynamical system. The initial states of the autonomous system are computed at reconfiguration (run-time), while the properties of this autonomous system can be computed design time, because the reference system introduced in Section 3.2 assumes design time knowledge of the system parameters. The ATSI lifetime timer on the figure specifies the duration of the anti transient signal.

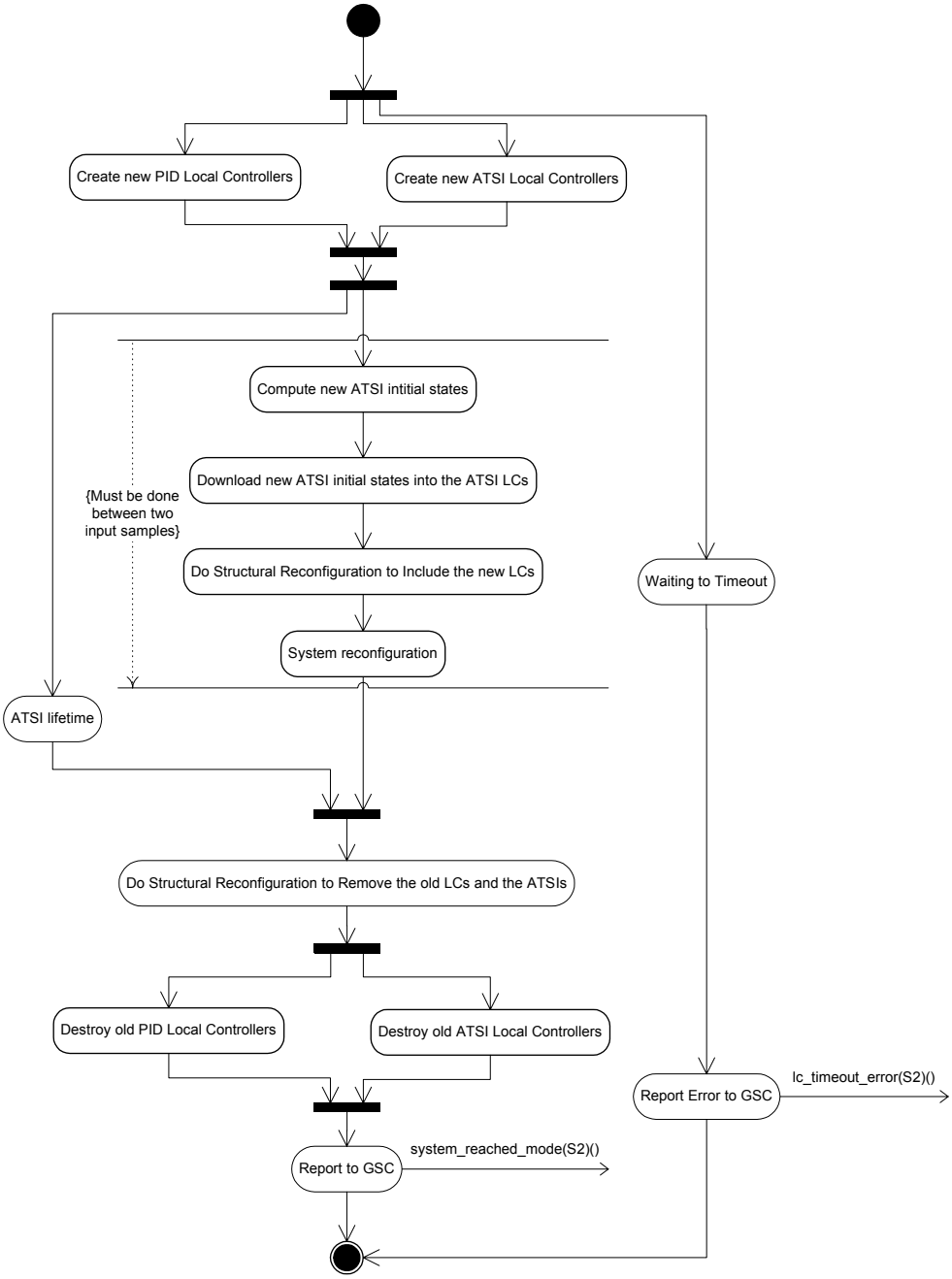

Fig. 20 Activity diagram of RM level post system change anti transient signal injection done during a reconfiguration from mode 1 to 2

9 Conclusions

The results of the investigations can be summarized in the following way:

One-step reconfiguration with state preservation using design-time information:

- Computational complexity and overhead of the reconfiguration method: Very low
- Storage requirements of the reconfiguration method: Pre-designed configurations must be stored in the regulators; therefore, storage requirements grow linearly with the number of configurations.
- Transient management: Design-time activity, no run-time resources are required.

One-step reconfiguration with state preservation using run-time design:

- Computational complexity and overhead: Very low
- Storage requirements: The current and the next configurations need to be stored in the regulators.
- Computational complexity and storage requirements of the run-time design depend on the regulators and their design algorithms. There are various methods to reduce computational complexity by increasing storage requirements, and/or by sacrificing precision.
- Transient management: Design-time activity, run-time resources are not needed.

Multiple-step reconfiguration on the LC level

- Computational complexity and overhead: Very low
- Storage requirements: At least three configurations need to be stored in the LC for linear interpolation, which are the old, the new, and the temporary configurations.
- Computational complexity and storage requirements of the run-time design depend on the regulators and their design algorithms used to work out the new and temporary configurations. Simple interpolation schemes guarantee *O(N)* complexity while more complex schemes may have unpredictable timing behavior.
- Transient management: Transient management is based on the design of the temporary configurations, so the design related statements apply here.

Multiple-step reconfiguration on the RM level

- Computational complexity and overhead on the LC level: See the one-step reconfiguration with state preservation using run-time design.
- Storage requirements on the LC level: See the one-step reconfiguration with state preservation using run-time design.
- Computational complexity and storage requirements of the run-time design depend on the regulators and their design algorithms used to work out the new and temporary

configurations. There are various methods to reduce computational complexity by increasing storage requirements, and/or by sacrificing precision.

- Transient management: Transient management is based on the design of temporary configurations, so the design related statements apply here.

Blending on the LC or on the RM level

- Computational complexity and overhead: At least doubled, because the old and the new controller should be operated in parallel.
- Storage requirements: At least doubled, because the old, and the new controllers and their data need to be stored.
- Computational complexity and storage requirements of the run-time design depend on the regulators and their design algorithms used to work out the new configurations. There are various methods to reduce computational complexity by increasing storage requirements, and/or by sacrificing precision.
- Transient management: Transient management is done by operating the old and the new systems in parallel, and the output is formed as a linear combination of the two. The overhead related to transient management is minimum.

Computation of the initial states (LC level only)

- Pure transient management method.
- Computational complexity and overhead of the transient management:
    - Instantaneous computation: high, it can be even unpredictable in time,
    - Filtering scheme: moderate, but it delays reconfiguration.
- Storage requirements of the transient management method: Strongly implementation dependent. The past inputs, outputs, or states may need to be stored to compute the initial states.

Anti-transient signal injection (RM level only)

- Pure transient management method.
- May require access to the internal states of the LC.
- Computational complexity and overhead of the transient management:
    - Intentional reconfiguration between controllers known in design-time: minimal at reconfiguration, the ATSI must be operated after the reconfiguration,
    - General case: high, and even unpredictable in time at reconfiguration; the ATSI system must be also operated.
- Storage requirements of the transient management method: Strongly implementation dependent. The past inputs, outputs, or states may need to be stored to compute the injected signal.

This summary clearly shows that the selection of proper reconfiguration and transient management method is a complex task, which requires detailed knowledge of the system in hand, the realization platform, and its real-time requirements. Furthermore, to achieve better transient properties higher computational power and more stringent real-time behavior is required. Therefore, a delicate balance needs to be found during the design.

References

[FWWW] http://www.isis.vanderbuilt.edu/Projects/Fact/Fact.htm

[HAR87] Harel, D., "Statecharts: A Visual Formalism for Complex Systems", *Science of Computer Programming 8*, pp. 231-274, 1987.

[KP2001] Kovácsházy T., Péceli, G., "Transients in Reconfigurable Signal Processing Channels", *IEEE Transactions on Instrumentation and Measurement*, Vol. 50, pp. 936-940, Aug. 2001.

[MR76] Mullis, C.T., R.A. Roberts, "Synthesis of minimum roundoff noise fixed point digital filters," *IEEE Trans. Circuits & Systems*, Vol. CAS-23, pp. 551-562, Sept. 1976.

[PK99] Péceli, G., Kovácsházy T., "Transients in Reconfigurable Digital Signal Processing Systems," *IEEE Transactions on Instrumentation and Measurement*, Vol. 48, No. 5, pp. 986-989, Oct. 1999.

[SKP02] Simon Gy., Kovácsházy T., Péceli, G., "Transient Management in Reconfigurable Control Systems", *BME-MIT Technical Report*, 2002.
Available at: http://www.mit.bme.hu/~khazy/publist.html

[PMP96] Padmanabhan, M., Martin K., and Péceli G., *Feedback-based Orthogonal Filters: Theory, Applications, and Implementation.* Kluwer Academic Publishers, Boston-Dordrecht-London, 1996. 265 p.

[RTUML] Douglass B. P., *Real-time UML : developing efficient objects for embedded systems, 2nd Edition.* Addison-Wesley, Upper Saddle River, 2000.