International Journal of Computer and Electrical Engineering, Vol. 3, No. 2, April, 2011
1793-8163

A General Concurrent Sequential Control System Supported by a Lattice Shaped Calculation Engine

JIN Shu, ZHOU Jin-guo, and YU Qi-hui

***Abstract*—To enhance the concurrency of the traditional sequential/program control systems, which adopt naive linear sequential ways of control step transition, the Domino sequential control system supports the simultaneous triggering of multiple control sequences and steps. Moreover, as the solution for the control flow continuation when a step failure is encountered, a two-way conditional transition mechanism is implemented through performing control steps bypathing. With the cooperation of the script engine embedded and a GUI configurator, Domino empowers users to customize control sequences to service all sorts of complicated control requirements proposed in the engineering of industrial automation systems. Furthermore, to meet with the security demand enforced by mission critical applications, a two-layer precondition validation procedure is dedicated for the recognizing and prevention of user operation faults. The performance, reliability, flexibility and configurability of the reference Domino implementation are supported by simulation results and real-world operations.**

***Index Terms*—Domino, Sequential Control, Programmed Control, Remote Control, Concurrent Operation, SCADA**

I. Introduction

With the wide deployment of PLCs and all sorts of on-site intelligent controllers equipped with micro-processors, industrial automation systems grow more and more sophisticated while the demand of human intervention is greatly reduced by contrast. However, the mission-critical operations such as to start/stop a generator, to energize/de-energize a traction power substation or to trigger an emergency mode for evacuation in a subway BAS (Building Automation System) subsystem remained under intense human supervision as ever before. All these operations share some characteristics in common: (1) each of them is composed of a set of homogeneous substeps; (2) the substeps are executed sequentially/concurrently in a deterministic way; (3) the operations integrated must be highly reliable. To perform a substep, some communication protocols supplied with two-phase interactions (send the command; wait for an ACK signal from the actuator and then execute the command), such as IEC870-5-101/103/104, are introduced by typical industrial automation systems to issue commands remotely in a reliable way. Organized in a sequential way, a set of remote control substeps may constitute a control sequence, which will be managed by the sequence/program control system.

Almost every decent SCADA (Supervisory Control And Data Acquisition System) system is furnished with a sequential control subsystem/component, so as to introduce reliability and simplicity into mission-critical user operations. As according to the manner of control sequence triggering, existing sequential control systems can be classified into two categories: the manually triggered, and the automatically triggered which may be activated by timers periodically or by specific events [1]. The sequential control logic can be executed in the backend control center, the substation and on-site local, thus constitutes another classification criterion [2]. In the past two decades, research and development efforts have been elaborated on every aspect of the sequential control system. LUO, DING et al discussed fault prevention mechanisms through precondition validation and emergency pause/stop supporting [2,3]; HAN, S. Malayappan et al presented detailed classification of early electro-mechanical and computerized implementation of sequential control systems and circuits hard-wiring respectively [4,5]; object-oriented way of control sequence flow mapping was introduced in late 90s [6-8]; Il Moon et al gave a formal definition of a model-based sequential control logic [9], while most sequential control systems in use apply LD/SD/SFC as the definition language/scheme [10-12].

Implemented with the API framework of the high performance real-time database platform of ChRDB [13], a novel sequence/programmed control system named Domino is presented. The system adopts a holistic approach to problem definition and solving so as to provide a generic tool applicable to all kinds of complicated integrated industrial automation operations. Compared with traditional sequential control systems, which are dedicated only to the execution of remote control operations, Domino offers a script engine that enables users to customize every piece of control logic in the control steps through supplying several lines of code. Moreover, with DomonoConfig - an application with intuitive graphical user interface provided, the system shows great configurability. To enhance fault-prevention, Domino follows a two-layer precondition validation procedure: first traverse all the global precondition steps to decide whether the control sequence can be executed or not; then before entering each control step activated, the corresponding preconditions are revalidated to make the final decision whether the execution of the step can be proceeded. Domino distinguished itself by the lattice-shaped control step interconnection that capacitates the triggering of multiple control steps in a single script command and the simultaneous execution of

Manuscript received December 23, 2010
JIN Shu, Railway-Transit Dept, SAC Research, Guodian Nanjing Automation Co., Ltd, Nanjing, P.R.China,(e-mail: jinshu@sac-china.com)
ZHOU Jin-guo, Railway-Transit Dept, SAC Research, Guodian Nanjing Automation Co., Ltd, Nanjing, P.R.China, (e-mail:ahzhjg@gmail.com)
YU Qi-hui, Railway-Transit Engineering Company, Guodian Nanjing Automation Co., Ltd, Nanjing, P.R.China, (e-mail:yuqihuiad@163.com).

multiple start steps. The support of two-way conditional transition is another advantage over its counterparts, which enable the control flow(s) to continue even after some step fails or timed out, through following the transition directives specified in the bypath configuration of the very control step.

II. Assumptions

Domino sequential control system is implemented as a major functional component of the DSC-9000W/U SCADA platform, which is created based on the in-memory real-time database named ChRDB [13]. Designed following an object-oriented scheme, ChRDB manages all the data/information as object instances that are derived from the specific templates configured in the RDBMS through a proprietary relation-object mapping mechanism. In the memory object pool, instances of data objects are broken into smaller entities - the attributes, i.e. an object instance is considered a list of attribute instances, which makes it easier to perform operations such as memory allocation, item indexing, sorting, and thus greatly improves the overall performance of data accessing and manipulation. As every object instance reside in the memory data pool is assigned an globally unique ObjectId, simply put the ObjectId of the target object instances into a link type attribute of the source object may establish a connection between the two object instances. With the object instance linking mechanism, typical data organization structures such as sequence/list, tree and net can be established.

To use with ChRDB, an application should firstly register itself as a client, then it may issue read/write operations to manipulate the object instances authorized and waiting for data write/change and configuration update events triggered by other ChRDB centered processes in cooperation. In view of the interaction between ChRDB and application processes, ChRDB may be considered an efficient IPC (Inter-Process Communication) mechanism.

The Domino daemon shares the same behavior with other ChRDB-centered applications: after registered with ChRDB, configuration information of all the control sequences are loaded and preprocessed, then the program start waiting for the triggering signals (ChRDB data write/change events) of control sequences validated and execute the control sequences activated as according to the Domino algorithm, which makes the following two assumptions:

A. The control sequences are deterministic. Any control sequence must contain a finite number of steps to transit to and thus all the possible transitions in any control step are the apriori knowledge;

B. No-Loop (use repeat/retry instead). The transition to any steps previously activated is disallowed and such logic should be replaced with self repetition or loop expansion through duplicating the target sub-branch.

III. Design and Algorithms

A. Definitions

Definition 1. Control Step := [Precondition; Action; Transition; Timeout;Bypath; State; MaxRetryTimes; Retries].

As described in Definition 1, the control step is defined as a 8-tuple: *Precondition* specifies the prerequisite conditions to enter the control step; *Action* is performed as a single-shot operation right after the *Precondition* passed validation; *Transition* constitutes the major part of the execution of a control step, which will be executed by the Domino engine repeatedly to check whether a specific condition is fulfilled and then make the corresponding transition(s) scheduled within the period set by *Timeout*; If a timeout is encountered in the continuous scanning/executing of the *Transition* scripts, the code supplied within *Bypath* is executed, which make it possible to continue and branch the control flow when no transition condition is met; in the life-time, status of the step execution is stored in *State* in real-time, here is the list of states supported: READY (the control step is just activated), PASSED (set after *Precondition* validated), EXECUTING (*Precondition* validated and checking conditions for possible transitions), TIMEOUT (timeout before making any transition), STOPPED (set when the control step stopped, timed out with empty *Bypath* configuration specified, force stopped or after a successful transition that automatically terminates the initiator step); once set, parameter *MaxRetryTimes* dictates the maximum times if the control step is re-triggered by a retry operation while *Retries* the counter of executions recorded up to present. As a major advantage over its counterparts, the Domino sequential control engine empowers users to write customized action scripts into individual fields of *Precondition*, *Action*, *Transition* and *Bypath*, which greatly promote the generality and flexibility of the sequential control system.

Definition 2. Control Sequence := ΣSi (IsControlStep(Si) ^ Sj, IsStartingControlStep(Sj))

The control sequence is defined as a finite set of control steps with at least one of them designated as a starting step. Traversing from the starting step(s), every other control step must be reachable, i.e. there exisits a transition path from one (or more, for possible joints) of the starting step(s) directing to every non-starting control step.

B. Algorithms

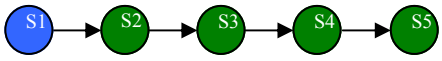


Fig 1. A Simple Control Sequence

Presented in Fig 1 is a typical control sequence containing five control steps. Once triggered, the staring control step S1 is activated and other control steps will follow in succession if and only if their predecessor succeeded execution and the corresponding transition conditions are fulfilled.

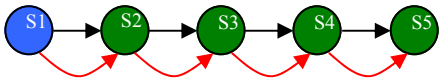


Fig 2. A Typical Control Sequence

Fig 2 depicts a typical control sequence applied in traction power supplying and distribution SCADA systems, which contains a list of control steps in succession and demands to keep the control flow through bypathing transitions even if some control steps failed to be fulfilled.

For abstraction, the problem of sequential/programmed

control can be remodeled as a set of transition-connected steps, in which diagram every node stands for a control step while the edges the valid transitions from source control steps to a target control steps.

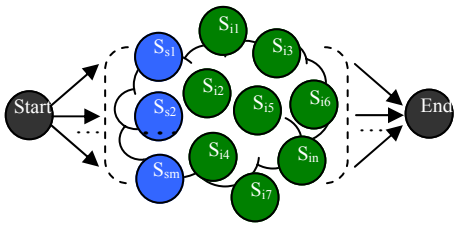


Fig 3. The Virtual Steps of "Start" and "End"

For the fact that the Domino daemon is waiting for the triggering of individual control sequences, a virtual common starting step named "Start" can be assumed as a part of the engine; likewise, when any control sequence stops or force stopped, its status will be collected correspondingly by the daemon, which procedure may be regarded as a virtual common final control step named "End". With the two virtual control steps presumed, together with the requirement that no loops and back trace linking is allowed, any valid control sequence configured according to the guidelines suggested by the Domino algorithms will take on a lattice shape, which means that any subset of control steps shares a common minimum upper bound and a maximum lower bound of control steps included in the control sequence. Fig 3 gives an abstraction for this scheme.

Defined as an 8-tuple, a control step is directly mapped into an object definition template with equivalent number of properties. Algorithm 1 describes in detail the execution of a control step triggered: Before entering a control step, the state ($S_{Current}.State$) and retry time count ($S_{Current}.Retries$) is initialized. Then the Domino engine will execute the precondition script ($S_{Current}.Precondition$) repeatedly waiting for the fulfillment of the preconditions within the time span specified by $S_{Current}.Timeout$. After the preconditions fulfilled, the scripts supplied with $S_{Current}.Precondition$ may perform a "proceed()" operation as described in Algorithm 2 to modify the execution state of the current control step to PASSED so as to trigger the code specified in $S_{Current}.Action$, otherwise the control step is terminated by automatically by Domino. Similarly, the Domino engine will then execute and check the transition script lines trying to make transitions. If timed out in the stage, script code customized in $S_{Current}.Bypath$ will be invoked for the support of control flow bypathing. In the situation when no transitions are successfully performed even after the execution of $S_{Current}.Bypath$, the Domino engine will check to see whether a retry is expected and make the control flow back-jump correspondingly as directed.

Algorithm 1. *Start a Control Step*

```
LABEL_START:
  S_Current.Retries ⟵ 1
  S_Current.State ⟵ READY
  T_start ⟵ CurTime
  While ((CurTime - T_start) < S_Current.Timeout) AND
          (S_Current.State = READY)
    Do S_Current.Precondition
  End
  If S_Current.State = PASSED Then
    S_Current.State ⟵ EXECUTING
    Do S_Current.Action
    If S_Current.State = EXECUTING Then
      T_start ⟵ CurTime
      While ((CurTime - T_start) < S_i.Timeout) AND
              (S_Current.State = EXECUTING)
        Do S_Current.Transition
      End
      S_Current.State ⟵ TIMEOUT
      Do S_Current.Bypath
    End
  End
  If S_Current.Retries == S_Current.MaxRetryTimes Then
    Stop S_Current
  Else
    S_Current.Retries ⟵ S_Current.Retries + 1
    Goto LABEL_START
  End
```

As described in Algorithm 2, 3 and 4, the operations of "proceed()", "retry()" and "stop()" involve simply the setting of the real-time status of the initiator control step, which is used as an intermediary signaling mechanism to coordinate the execution control flow of the current control step.

Algorithm 2. *Proceed Operation*

```
S_Current.State ⟵ PASSED
```

Algorithm 3. *Retry Operation*

```
S_Current.MaxRetryTimes ⟵ N
```

Algorithm 4. *Stop a Control Step*

```
S_Current.State ⟵ STOPPED
```

In Domino sequential control system, step transition operations (invocations of the "step()" operation as described in Algorithm 5) are embedded in the user-customizable script code supplied with *Precondition*, *Action*, *Transition* or *Bypath*. Before making a transition to a target control step, the Domino engine will check whether it is already executed so as to filter-out possible loops; if not, the target control step(s) will be push onto the stack of steps executed denoted as *Ue* and then be activated. In the single step mode of manually triggered execution, only the first target step S_1 will be stacked, marked and delayed to be executed when a next sequence triggering signal is received.

Algorithm 5. *Step Transition*

```
S ⟵ Control Steps to start
For each control step S_i in S
  If S_i ⊄ U_e Then
    Start S_i
    Add S_Target to U_e
  End
End
Stop S_Current
```

Domino introduces concurrency into the sequential control system through supporting multiple target control steps transition. As illustrated in Fig 4, S1 is a staring control step (painted with blue), which will make a transition to S2 (a common control step, painted with green) if the transition conditions are fulfilled; the latter will relay the control flow to trigger three control steps of S3, S4 and

S5 concurrently after its completion, and if not, the control flow is bypathed to S6. In either case, the control step of S7 is the next stop in the control sequence, which is regarded as a joint that collects execution results of all these precedent control steps for the making of the decisions of transitions followed. Supplied with adequate logics, S7 may act as a voter, a MUX, an adder or any other kind of logical operator.

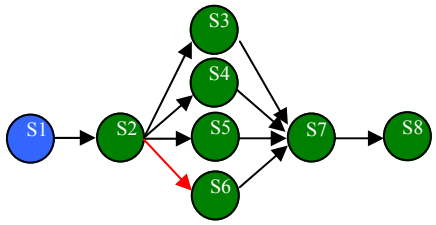


Fig 4. Concurrent Execution, Bypathing and Jointing

Described in Fig 5 is a c control sequence with multiple threads of execution. After step S2 is completed successfully, S3, S4, S5 are started simultaneously so as to separate the execution flow into three branches; if not, the control flow will be directed to the branch leading by S6. If and only if all the branches forked finished, a control sequence terminates.

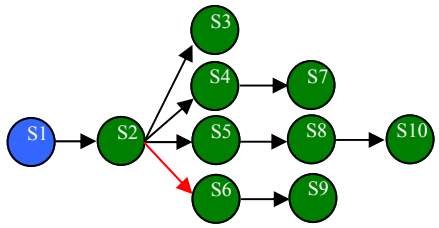


Fig 5. Control Sequence with Multiple Threads

Typically, a control sequence contains one staring control step as exampled in Fig 1-5. However, Domino makes a significant step forward; the system supports the concurrent activation of multiple starting control steps configured in one control sequence. As shown in Fig 6, the control sequence contains three starting control steps: S1, S2 and S3, which will be executed simultaneously when the control sequence is triggered. Algorithm 6 gives in detail of the execution of multiple staring steps, of which the triggering of a control sequence with single staring step is just a particular case.

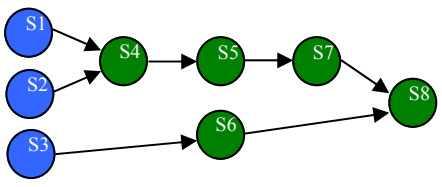


Fig 6. Multiple Starting Steps

Algorithm 6. *Triggering a Single Control Sequence*

S ⟵ starting control steps of the sequence
For each step S_i in S
  Start S_i
End

Similar to the triggering of multiple staring steps, Domino even empowers the users to execute multiple control sequences simultaneously through performing a single "start()" operation. As depicted in Fig 7, S1 is the starting and the only control step of a control sequence, whose Action script is supplied with a line of "start(Seq1,Seq2…)." When triggered, Domino engine will assign each control sequence to be triggered an execution thread correspondingly and kick it start as described in Algorithm 7.

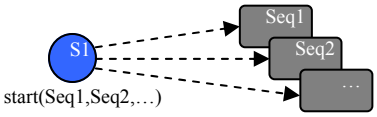


Fig 7. Concurrent Control Sequences Triggering

Algorithm 7. *Multiple Control Sequences Triggering*

S ⟵ Control sequences to start
For each sequence S_i in S
  Trigger S_i
End

Algorithm 8 gives the termination procedure of a control sequence, which is provided for the emergency stopping of a control sequence so as to prevent from misoperation or at least mitigate the damage caused. It's worth noted that after completion, a control step will be removed from the stack of control steps executed, so as to re-enable the very control step for further execution.

Algorithm 8. *Finish a Control Sequence*

S ⟵ Control steps in the sequence
For each control step S_i in S
  Stop S_i
  Remove S_i from U_e
End

IV. Simulations

Domino is implemented in the C++ programming language using QT 4.6.2 framework, all the components are compiled by Microsoft Visual C++ 2008 with the optimization option "/O2" turned on. On the test-bed computer (Lenovo ThinkPad SL410 2842-56C，Intel Core 2 Duo 2.2GHZ with 2G RAM), simulation tests (control sequences) are executed to obtain the time consumption of best-effort sequence execution and the maximum throughput of control steps execution, which adopt remote control operation as the directing logic of control steps. To simulate the real-world two-phase remote control interactions (although considerably faster), a piece of calculation service is dedicated to generate responses echoed from the actuator devices virtualized.

Fig 8 and Fig 9 show the unit execution time and the control steps execution throughput of a single sequence with 1 to 32 remote control steps respectively, which support the conclusion that when the number of control steps contained in a control sequence grows linearly, the execution time will responds with a quadratic growth. However, with threading overhead costs most of the CPU time, the throughput of control steps execution suffers only a linear loss when a control sequence put on more control steps. Fig 9 shows an exception when the step count is two, which can be considered an environment-specific parameter, figured out and may be used in the further optimization through

separating every control sequence into sub-sections with the ideal number of control steps.

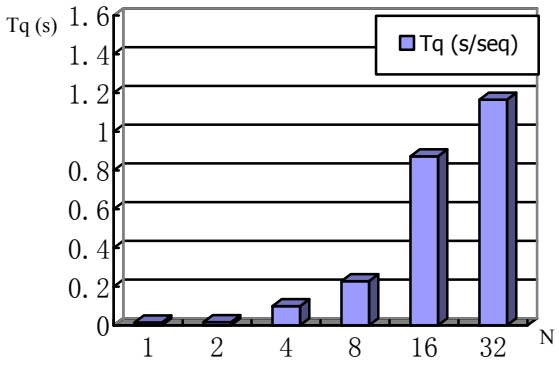


Fig 8. Sequence Execution Time (single sequence)

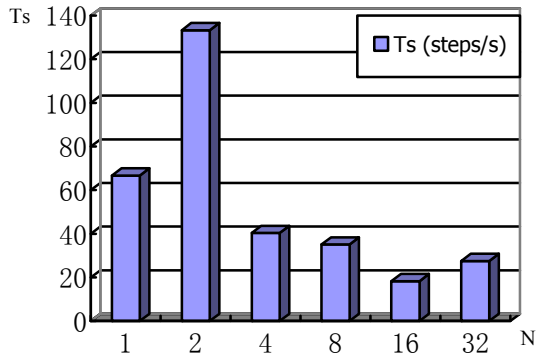


Fig 9. Steps Processing Throughput (single sequence)

Fig 10 and Fig 11 give the variation of the same variables mapped with the cooperation of different numbers of control steps (N) and control sequences (M), which derives the fact that the more the inter-process locking operations invocated, the less efficient the execution of concurrent control steps may be. The rule is applicable both to the number of control steps contained in a control sequence and the number of concurrent control sequences in execution. It's recommended to break up complex control sequences into trivial ones connected with a multiple-triggering control sequence. Prepared to be used in the operation of host performance self adaptation, the optimized values of the variables should be calculated in advance.

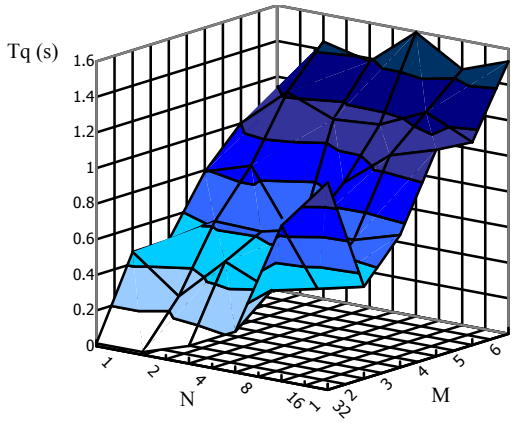


Fig 10. Sequence Execution Time (multiple sequences)

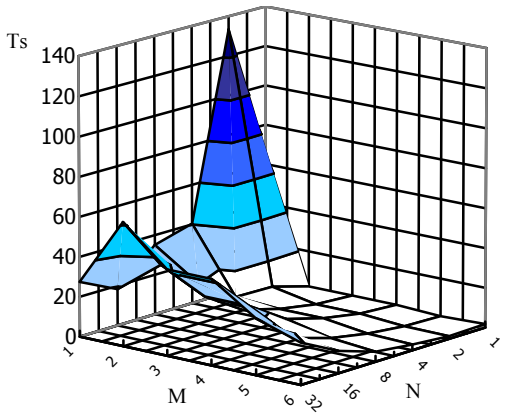


Fig 11. Steps Processing Throughput (multiple sequences)

V. Summary and the Future Work

Implemented as the sequential control component of DSC-9000 SCADA system [14], Domino is put to in-field operation in many projects domestic and abroad including Beijing subway line 1/2 PSCADA system, Teheran subway line 4 PSCADA system, Dazhou-Wanzhou electrical railway traction power supplying/dispatching system, etc. The system has successfully passed all these tough tests and proven to be highly configurable, reliable and efficient. A GUI utility named DominoConfig illustrated as Fig 12 is also developed to servicing the management and configuration of control sequences for the Domino sequential control system.

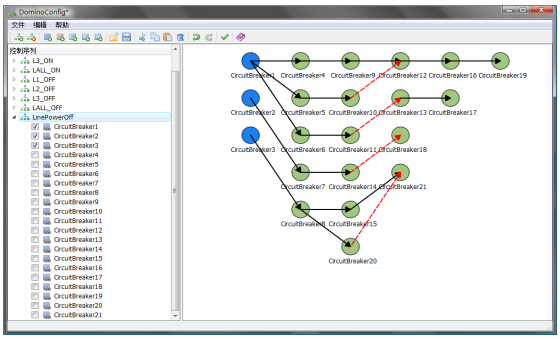


Fig 12. Screen-shot of DominoConfig

Recently a prototype version of the Domino system that supports all the POSIX-compliant UNIX variants (Sun Solaris, IBM AIX, HP UX, Linux) is implemented and under last stage stress testing. The focus of the next stage development work will be put on the following three topics: (1) direct utilization of the concurrent processing power supported by hardware newly emerged (SMP / Multi-Core); (2) HMI configuration scheme with more intelligence; (3) GUI application for automatic control sequence execution and status monitoring; (4) more advanced control sequences mapping and segmentation.

Acknowledgment

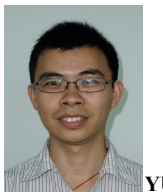
Many thanks go to Mr. LIU Zhi-chao for his invaluable guidance during the design and implementation of the Domino sequential/programmed control system. Thank you Mr. LIU Wei and JIA Zhi-min, without your encouragement and support, the work can not be accomplished as smoothly.

REFERENCES

[1] LI Fei, FENG Yong-qing, LIANG Shouyu, "Design of Sequence Control Software in China Southern Grid EMS," Automation of Electric Power Systems, NARI Press: Nanjing, vol 31, pp.194-196, Dec 2007.
[2] LUO Xiao-li, WU Wen-xia, GONG Yuan, "Sequential Operations in Cell Devices," Electric Power Automation Equipment, SAC Press: Nanjing, vol 28, pp.115-121, Jun 2008.
[3] DING Quan, ZHU Lai-qiang, HU Dao-xu, JIANG Yan-jun, JIANG Hui, "Programmed Operations of Substation and Execution by Telecontrol Device," Electric Power Automation Equipment, SAC: Nanjing, vol 27, pp.119-121, Aug 2007.
[4] HAN Zhong-xu, "Application of Sequence Control Technique in Power Plant and Discussion on Coordinated Development of Sequence Control Technique and Computer Control System," Power System Technology, SG Press:Beijing, vol 25, pp.63-68, Oct 2000.
[5] S. Malayappan, K. Arul Raj, S. Sathya Arunachalam, S. Venugopal, D. Ramalingam, "Design of a Sequential Control Circuit for An Industrial Robot Using Cascading Method," Proceedings of the International Conference on Man-Machine Systems, Batu Ferringhi, Penang, MALAYSIA, Oct 2009.
[6] Martinez X C P, Garcia R F, "SFC++: A Tolls For Developing Distributed Real-Time Control Software," Microprocessors and Microsystems, Elsevier: Amsterdam, vol 23, pp.75-84, Feb 1999.
[7] Chiaming Yen, Wu-Jeng Li, Jui-Cheng Lin, "A Web-based, Collaborative, Computer-Aided Sequential Control Design Tool.," Control Systems, IEEE: Piscataway, NJ, vol 23, pp.14-19, Apr 2003.
[8] HUANG Yong, LONG Hong-sheng, ZHUANG Cheng, "Object-Oriented Programming of Conveyors Sequential Control System.," Control Engineering of China, NEU Press:Dalian, vol 12, pp.53-56, Jan 2005.
[9] Il Moon, Gary J. Powers, Jerry R. Burch, Edmund M. Clarke, "Automatic Verification of Sequential Control Systems Using Temporal Logic," AIChE Journal, Wiley: Hoboken, NJ, vol 38, pp.67-75, Jan 1992.
[10] Mohammad Abu Zalata, Mohammad Alia, "Realizing Sequential Processes Using Programmable Logic Controllers (PLCS)," International Journal of Simulation, XXX:XXX, vol 5, pp.64-70, Sep 2004.
[11] Finn Haugen, "Article: Sequential Control," http://home.hit.no/~hansha/documents/lab/LabWork/SequentialControl/Background/sequential_ control.pdf. 2009.8.
[12] N. A. Ivanescu, Th. Borangiu, S. Brotac, A. Dogar, "Implementation of Sequential Function Charts with Microcontrollers," Proceedings of the 15th Mediterranean Conference on Control & Automation, Athens, Greece, Jul 2007.
[13] JIN Shu, DAI Hong-bin, JIA Zhi-min, "ChRDB : An Object-Oriented Real-Time Database With High Performance," Electric Power Automation Equipments, SAC Press: Nanjing, vol 29, pp.88-93, Dec 2009.
[14] JIN Shu, DAI Hong-bin, ZHU Chao, "DSC-9000 Automation Platform User's Manual," Techinical Report, Guodian Nanjing Automation Co.,LTD. 2010.

JIN Shu, born in 1979, received his Ph.D. degree in computer engineering in Nanjing University of Science and Technology in 2006. He is currently an ACM member and works for SAC Research of Guodian Nanjing Automation Co.,LTD. As a senior software engineer, he focuses his R&D interests on the design and implementation of SCADA systems and applications. He has published 13 papers on a wide range of topics such as SCADA system design, real-time databases, system performance tuning and information security in domestic and international journals & conferences during the last decade.

ZHOU Jin-guo, born in 1982, received his MS.c degree in computer engineering in the Computer Science Department of Central South University. He works for the Railway-Transit Dept of SAC Research, Guodian Nanjing Automation Co.,LTD. as a software engineer. Recently, he commits himself to the research and development work of an upgraded version of the existing sequential/programmed control system with better durability, capability and configurability.



YU Qi-hui, born in 1982, He is a system engineer of Nanjing SAC Railway-Transit Engineering Co.,LTD. As an experienced on-site problem solver, he devoted himself to the creation, configuration and optimization of various substation automation, traction power supplying supervision, and subway PSCADA systems for a straight 5 years. He currently works on the Domino sequential control system project as the major tester and requirement consultant.