Available online at www.sciencedirect.com
ScienceDirect
Future Generation Computer Systems 23 (2007) 804–815
FGCS FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

Coordinated access control with temporal and spatial constraints on mobile execution in coalition environments

Song Fu*, Cheng-Zhong Xu

Department of Electrical and Computer Engineering, Wayne State University, Detroit, MI 48202, United States

Received 11 April 2006; received in revised form 13 October 2006; accepted 11 December 2006
Available online 22 December 2006

Abstract

Dynamics is an inherent characteristic of computational grids. The volatile nodal availablity requires grid applications and services be adaptive to changes of the underlying grid topology. Mobile execution allows mobile users or tasks to relocate across different nodes in the grid. This poses new challenges to resource access control. Resource sharing in the grid coalition environment creates certain temporal and spatial requirements for accesses by mobile entities. However, there is a lack of formal treatment of the impact of mobility on the shared resource access control. In this paper, we formalize the mobile execution of grid entities by using the mobile code model. We introduce a shared resource access language, SRAL, to model the behaviors of mobile codes. SRAL is structured and composed so that the program of a mobile code can be constructed recursively from primitive accesses. We define the operational semantics of SRAL and prove that it is expressive enough for most resource access patterns. In particular, it is complete in the sense that it can specify any program of *regular trace model*. A constraint language, SRAC, is defined to specify spatial constraints for shared resource accesses. Checking if the behavior of a mobile code satisfies a given spatial constraint can be solved by a polynomial-time algorithm. We apply the Duration Calculus to express temporal constraints, and show the constraint satisfaction problem is decidable as well. We extend the role-based access control model to specify and enforce our spatio-temporal constraints. To prove the concept and technical feasibility of our coordinated access control model, we implemented it in a mobile agent system, which emulates mobile execution in grids by software agents.

Keywords: Coordinated access control; Temporal constraint; Spatial constraint; Mobile execution; Coalition environment; Computational grid

1. Introduction

Computational grids integrate geographically distributed resources into a seamless environment. By spreading workloads across a large number of computers, grid users can take advantage of enormous computational, storage, and bandwidth resources that would otherwise be prohibitively expensive to attain within traditional multiprocessor supercomputers. Recent grid computing efforts have included computation for the human genome project [3], climate change research [27], and earthquake engineering [29]. These grand applications utilize grid resources for high-throughput computations.

Dynamics is an inherent characteristic of computational grids. Nodal failures, overload and attacks make the grid topology volatile. Research in [24] shows that CPU availability and host availability in grids are subject to rapid change. Adaptivity is fundamental to achieving application performance and to enhancing the utilization of available resources in such dynamic environments [4].

To construct resilient services and to treat computers in a grid as a single, unified pool, several solutions have been proposed. They define execution units at different layers of the runtime system, encapsulating and migrating the states of these execution units across multiple computers. Mobile execution can be achieved by migrating entire operating systems [8,32], runtime supporting services [19], or application processes/threads [30,26], among grid nodes. In addition, with the emergence of powerful portable computing devices, along with advances in wireless communication technologies, there are recent research efforts on building mobile grids [7,31], which integrate mobile devices into computational grids. This work enriches the semantics of mobile execution in grids.

* Corresponding author. Tel.: +1 313 577 5147.
E-mail addresses: song@eng.wayne.edu (S. Fu), czxu@eng.wayne.edu (C.-Z. Xu).

0167-739X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2006.12.002

A mobile entity, i.e. either a mobile user or a mobile execution unit, may relocate between different networks and connect to different data servers at different times. The mobile execution environment no longer requires a mobile entity to be maintained at a fixed and universally known position in the grid and it enables unrestricted mobility of users/tasks. In addition, multiple mobile entities may participate in a teamwork. These features of mobile execution create new challenges for resource access control, because permissions may be granted based not only on the requesting subjects, but also on the previous access actions of mobile entities and even of their companions. This requirement is frequently encountered in a coalition environment,[1] where the servers are generally cooperative and trustworthy, and they provide shared resources to outsiders.

In a coalition environment, the computation of a mobile entity may be spread across several hosting nodes and each access to the resources of a host lasts for a certain period of time. These spatio-temporal features pose a need for coordinated access control over shared resources at different hosts. Examples of such requirements are "if a mobile entity accesses a resource r, e.g. a licensed software package or its trial version, on host h_1 too many times during a certain time period, it is not allowed to access that resource on host h_2 forever" and "tasks of paid user Alice are allowed to use the CPU resources of a IBM SP-2 machine for 2 h". We note that these security requirements contain both temporal and spatial constraints over time-sensitive resource access activities on more than one host. They are beyond the capability of any of today's formal reasoning tools and access control models for mobile execution in grids.

In this paper, we present a logical framework to support a coordinated access control model enforcing both temporal and spatial constraints related to grid-based mobile execution. To tackle the security challenges introduced by execution mobility, we formalize the mobile execution in grids by using the mobile code model [20]. A mobile user roaming around networks or an execution unit migrating across grid computers is represented by a mobile code moving among different hosts and executing its programs on behalf of its owner. This framework takes the usage durations of permissions and access history information of mobile entities into account when granting permissions to mobile entities. Our spatio-temporal access control model includes a Shared Resource Access Language (SRAL) for the specification of access patterns of a mobile entity. The access language is structured and expressive enough to specify various access structures. Semantics of the language is established for spatial constraint satisfaction reasoning. SRAL is complete in the sense that it can specify any program of *regular trace model*. We note that the role-based access control approach [14] simplifies the permission management in grid-based mobile execution, the indirect assignment of permissions to subjects and the permission inheritance in role hierarchies facilitate the privilege delegation and security policy making. The constraint specification and enforcement mechanisms facilitate designing the access control schemes for mobile entities. Our coordinated access control model extends the basic role-based access control model to specify and enforce spatio-temporal constraints. A constraint language is defined to specify spatial constraints for accessing shared resources. The problem of checking if a mobile code program satisfies a given spatial constraint can be solved by a polynomial-time algorithm. We apply a continuous time model and boolean-valued state functions to express temporal constraints, and prove the constraint satisfaction problem is decidable. To prove the concept and technical feasibility of our coordinated access control model, we implement it in a mobile agent system, which emulates mobile execution in grids by software agents.

The rest of the paper is organized as follows. Section 2 describes the model for coalition mobile execution and assumptions. Section 3 presents spatial constraints for coordinated mobile access control. Temporal constraints and satisfaction checking are discussed in Section 4. Section 5 focuses on the implementation of our coordinated access control model in the Naplet mobile agent system. In Section 6, we present related work. Section 7 concludes the paper with remarks on future work.

2. A coalition mobile execution model and assumptions

A mobile code[2] may roam across networks to perform computation. At each host, the mobile code executes operations with access to resources of different types. Since private resources of a host can be protected under local control, we focus on accesses to shared resources among hosts.

A "coalition environment" could be commercial, in which corporations form a partnership, or governmental/military, in which several nations work together to achieve a common goal. In either case, the defining characteristic is the presence of multiple organizations or entities that are unwilling to rely on a third party to administer trust relationships. Consequently, the organizations must cooperate to share the subset of their protected resources necessary to the coalition, while protecting the resources that they don't want to share. To model a generic coalition environment, specify the coordinated access control scheme and consider its properties, we define the following syntactic sets:

- H: a set of hosts. Let $h, h_1, h_2, \ldots$, range over H.
- R: a set of shared resources. Let $r, r_1, r_2, \ldots$, range over R.
- OP: a set of operations on the shared resources R. Let $op, op_1, op_2, \ldots$, range over OP, such as *execute, read, write* for the file system related applications.
- Z: a set of channels. Let $\alpha, \beta, \gamma, \ldots$, range over Z.
- V: a set of variables. Let $v, v_1, v_2, \ldots$, range over V.
- C: a set of boolean or condition expressions. Let $c, c_1, c_2, \ldots$, range over C.

[1] The computational grid is a coalition environment, where companies and research institutes cooperate to share a subset of their protected resources necessary to run high-performance, high-throughput applications, while protecting the resources that they do not want to share.

[2] We use the mobile code computational diagram to model the behaviors of mobile users and migrating execution units in grids. Therefore, we only consider access control for mobile codes in the following discussion.

- E: a set of arithmetic expressions. Let $e, e_1, e_2, \ldots$, range over E.
- Ξ: a set of signals. Let $\xi, \xi_1, \xi_2, \ldots$, range over Ξ.

A mobile code makes accesses to the shared resources on one host and then moves to another. It also needs to exercise input and output through the communication channels, synchronize its actions with others in a teamwork. Its program includes the instructions for shared resource accesses and synchronization operations. An access is represented by a tuple $a = (s, op, r, h)$, which means that the mobile code subject s exercises operation op on resource r at host h. It is the consequence of executing an instruction of the mobile code's program, and at the same time the corresponding permissions are granted by the access control mechanism. The set of all accesses by a mobile code is defined as $A = \{(s, op, r, h)\}$. We assume when an access request to a shared resource is executed by a coalition host that a *execution proof* will be issued for the mobile code. This records the information of (s, op, r, h) of the access, and a timestamp. So the semantics of an execution proof, denoted by $Pr_x(\cdot)$, is that for an access a to a shared resource accessed by a mobile code, $Pr_x(a) = true$ iff access a has been successfully carried out at host $a.h$.

3. Spatial constraints for coordinated access control

In mobile execution, a mobile code roams cross a network carrying out its computation on coalition hosts according to its program. This motivates us to define a shared resource access language to specify the access behavior that a mobile code will make during its execution, and to construct a logical framework to reason about its safety properties.

3.1. Syntax of shared resource access language

Intuitively, the program of a mobile code specifies the access actions that it intends to perform and their ordering. For example, a mobile code s reads socket resources $r_1, \ldots, r_n$ in that order; or, it might read r_1 first and then, if variable $v > 0$ then write r_2 else write r_3. We argue that a shared resource access language and its semantics should be expressive enough so that various access patterns can be specified. The access language should also be structured and composed so that a mobile code program can be constructed from primitive or other composite programs. This also facilitates formalizing the semantics of the language.

Inspired by the constructs in current programming languages such as C++ and Java, we define a shared resource access language called SRAL.

Definition 3.1 (*Shared Resource Access Language*). Accesses by a mobile code to the shared resources are defined as follows using the BNF notation:

$$a ::= op\ r\ @h \mid \alpha?v \mid \alpha!e \mid signal(\xi) \mid wait(\xi) \mid a_1; a_2 \mid if\ c\ then\ a_1\ else\ a_2 \mid while\ c\ do\ a$$

where the symbol "::=" should be read as "is defined as", the symbol "|" as "or" and the symbol "@" as "at". □

We explain each construct intuitively as follows. A single resource access is represented by the primitive $op\ r\ @\ h$ which denotes an operation op on a shared resource r at host h. $\alpha?v$ will receive a value from channel α and assigns the value to variable v. If channel α is empty, then the mobile code instance will wait over the channel. $\alpha!e$ will append the value of arithmetic expression e to channel α. If α is originally empty, then in addition, it will wake up all the mobile code instances that wait on it. $Signal(\xi)$ and $wait(\xi)$ will perform an order synchronization such that $signal(\xi)$ has to be performed first before $wait(\xi)$ can be performed. $a_1; a_2$ is the sequential composition of accesses a_1 and a_2. *if c then* a_1 *else* a_2 is a conditional composition of a_1 and a_2. In particular, if condition c evaluates to true, then a_1 will be carried out, otherwise, a_2 will be carried out. *while c do a* will carry out access a whenever c is still true.

3.2. Operational semantics of SRAL

In the following, we will define the operational semantics of SRAL in terms of a set of inference rules. Using this set of inference rules, one can derive the semantics of executing the program of a mobile code. First, we introduce several notations:

- Σ: the set of states of a mobile code system. The definition of what counts as a state is not important to us, i.e., any notion of a state is allowed here. Let $\sigma, \sigma_1, \sigma_2, \ldots$, range over Σ.
- The evaluation of a condition c in a state σ is defined by function: δ: $C \times \Sigma \rightarrow \{true, false\}$. Therefore, $\delta(c, \sigma)$ represents the truth value of c in state σ.
- S: a set of mobile codes in the system. Let $s, s_1, s_2, \ldots$, range over S.
- A: a set of accesses to shared resources by mobile codes in a coalition system. Let $a, a_1, a_2, \ldots$, range over A.
- The semantics of a mobile code accessing a shared resource is given by function $\rho : S \times A \times \Sigma \rightarrow \Sigma$. After mobile code s performs an access a starting at state σ, the resulting state will be $\rho(s, a, \sigma)$.[3] Function ρ models the semantics of a mobile code's action at a particular host. This semantics can be derived from the definition of conventional programming languages that are used to specify a mobile code's action.
- $X ::= 2^{S \times A \times P}$, the set of *execution* of a system. Each execution is a set of tuples (s_j, a_j, p_j) $(s_j \in S, a_j \in A, p_j \in P, j = 1, \ldots)$ that specifies a mobile code s_j performing access a_j to a shared resource at a host with a program p_j. Let $x, x_1, \ldots$, range over X. Note that X is infinite since P is infinite.
- $\Omega ::= \Sigma \times X$, the set of *configurations* of a system. Each configuration is a pair (σ, x) where $\sigma \in \Sigma$ and $x \in X$. *Ω defines the state space of the whole system.* It is infinite since X is infinite. However, it will be shown that given an initial configuration, a system can only *reach* a finite set of configurations. Only this set of reachable configurations are our concern for a particular system.

The set of inference rules that defines the operational semantics of SRAL is formalized in Fig. 1. We explain each of them in the following:

[3] We abbreviate it by $\rho(a, \sigma)$ when s is understood from the context.

1. $\dfrac{}{(\sigma,\{(s,a,a),\ldots\}) \rightarrow_a (\rho(s,a,\sigma),\{\cdots\})}$
2. $\dfrac{}{(\sigma,\{(s,\varepsilon,a),\ldots\}) \rightarrow_a (\sigma,\{(s,a,a),\ldots\})}$
3. $\dfrac{a_1 \neq a_2}{(\sigma,\{(s,a_1,a_2),\ldots\}) \rightarrow_{a_1} (\sigma,\{(s,a_2,a_2),\ldots\})}$
4. $\dfrac{(\sigma,\{(s,a,p_1),\ldots\}) \rightarrow_\lambda (\sigma',\{(s,a',p_1'),\ldots\})}{(\sigma,\{(s,a,p_1;p_2),\ldots\}) \rightarrow_\lambda (\sigma',\{(s,a',p_1';p_2),\ldots\})}$
5. $\dfrac{}{(\sigma,\{(s,\varepsilon,a;p),\ldots\}) \rightarrow_a (\sigma,\{(s,a,a;p),\ldots\})}$
6. $\dfrac{head(\alpha) = n}{(\sigma,\{(s,a,\alpha?v),\ldots\}) \rightarrow_{\alpha?n} (\sigma[n/v][tail(\alpha)/\alpha],\{\cdots\})}$
7. $\dfrac{}{(\sigma,\{(s,a,\alpha!e),\ldots\}) \rightarrow_{\alpha!\sigma(e)} (\sigma[\alpha@[\sigma(e)]/\alpha],\{\cdots\})}$
8. $\dfrac{}{(\sigma,\{(s,a,signal(\xi)),\ldots\}) \rightarrow_{signal(\xi)} (\sigma[\Gamma\cup\{\xi\}/\Gamma],\{\cdots\})}$
9. $\dfrac{\xi \in \Gamma}{(\sigma,\{(s,a,wait(\xi)),\ldots\}) \rightarrow_{wait(\xi)} (\sigma[\Gamma-\{\xi\}/\Gamma],\{\cdots\})}$
10. $\dfrac{\delta(c,\sigma) = true}{(\sigma,\{(s,a,if\ c\ then\ p_1\ else\ p_2),\ldots\}) \rightarrow_\tau (\sigma,\{(s,a,p_1),\ldots\})}$
11. $\dfrac{}{(\sigma,\{(s,a,p_1 \| p_2),\ldots\}) \rightarrow_\tau (\sigma,\{(s,a,(p_1;signal(\xi)) \| (p_2;wait(\xi))),\ldots\})}$
12. $\dfrac{(\sigma,\{(s,a,p_1),\ldots\}) \rightarrow_\lambda (\sigma,\{(s,a,p_{11} \parallel p_{12}),\ldots\})}{(\sigma,\{(s,a,p_1;p_2),\ldots\}) \rightarrow_\lambda (\sigma,\{(s,a,p_{11} \parallel (p_{12};p_2)),\ldots\})}$
13. $\dfrac{}{(\sigma,\{(s,a,p_1 \parallel p_2),\ldots\}) \rightarrow_\tau (\sigma,\{(s,a,p_1),(s',a,p_2),\ldots\})}$
14. $\dfrac{\delta(c,\sigma) = false}{(\sigma,\{(s,a,if\ c\ then\ p_1\ else\ p_2),\ldots\}) \rightarrow_\tau (\sigma,\{(s,a,p_2),\ldots\})}$
15. $\dfrac{\delta(c,\sigma) = true}{(\sigma,\{(s,a,while\ c\ do\ p),\ldots\}) \rightarrow_\tau (\sigma,\{(s,a,p;while\ c\ do\ p),\ldots\})}$
16. $\dfrac{\delta(c,\sigma) = false}{(\sigma,\{(s,a,while\ c\ do\ p),\ldots\}) \rightarrow_\tau (\sigma,\{\cdots\})}$

Fig. 1. Operational semantics of SRAL.

- Rule 1 describes the semantics of *executing* a primitive access a by mobile code s. Intuitively, it states that "after the execution of primitive access a to a shared resource on a host, the state component is transformed into $\rho(s, a, \sigma)$, and (s, a, a) will be deleted from the execution component".
- Rules 2 and 3 describe the semantics of *attempting to execute* a primitive access by mobile code s. The former is to process the mobile code program from the beginning, where the currently-performed access is ε, i.e. empty. The latter models the semantics of execution of a mobile code: s performing access a_1 originally, to execute primitive access a_2 where $a_1 \neq a_2$, s has to perform a_2, which results in the change of the configuration to one that reflects the new execution trace of mobile code s while the state component remains unchanged.
- Rule 4 describes the semantics of executing a sequential composition of two sub-programs $p_1; p_2$ by mobile code s. It says that "if there exists a configuration transition such that σ is transformed to σ', and execution $\{(s, a, p_1), \ldots\}$ is transformed to $\{(s, a', p_1'), \ldots, \}$ (specified in the hypothesis), then there exists a configuration transition such that σ is transformed to σ', and $\{(s, a, p_1; p_2), \ldots\}$ is transformed to $\{(s, a', p_1'; p_2), \ldots\}$.
- Rule 5 describes the semantics of executing a composition program of mobile code s from the start. The program includes a primitive access a sequentially followed by the remaining part. The inference rule says that access a will be first performed by mobile code s, which will eventually change the state component according to the above rules, and then the remaining program p will be executed.
- Rule 6 describes the semantics of receiving a value n from channel α when mobile code s executes access $\alpha?v$ on a host. The hypothesis specifies that the head of channel α is value n, and the conclusion of the rule specifies that the state of the system will be transformed into one in which v is assigned with value n and the head element n is deleted from channel α.
- Rule 7 describes the semantics of sending arithmetic expression e to channel α when mobile code s executes access $\alpha!e$ on a host. The hypothesis is empty, and the conclusion of the rule specifies that $\sigma(e)$, the evaluation of arithmetic expression e in state σ, is appended to the end of channel α. Rules 6 and 7 are applied to specify the mobile code communication in a teamwork.
- Rule 8 describes the semantics of sending signal ξ to signal pool Γ. The hypothesis is empty, and the conclusion of the rule specifies that signal ξ is added into signal pool Γ.
- Rule 9 describes the semantics of receiving signal ξ from signal pool Γ. The hypothesis specifies that signal ξ must be present in the signal pool Γ, and the conclusion of the rule specifies that signal ξ is added into signal pool Γ. Rules 8 and 9 are applied to specify the synchronization among mobile codes in a teamwork.
- Rules 10 and 14 describe the semantics of executing a conditional program *if c then* p_1 *else* p_2 by mobile code s when c is evaluated to be true ($\delta(c, \sigma) = true$) or false ($\delta(c, \sigma) = false$), respectively.
- Rules 11 and 12 describe the semantics of operator $\|$ in terms of the semantics of a new operator $|||$, $signal(\xi)$ and $wait(\xi)$. The semantics of $|||$ is described next.
- Rule 13 describes the semantics of the parallel operator $|||$. It models situations when several mobile codes perform parallel tasks in a teamwork. Intuitively, a teammate of s is made, and in the resulting configuration, mobile code s is associated with p_1, and its teammate is associated with p_2.

  Intuitively, the difference between $|||$ and $\|$ is that synchronization is explicit for $|||$ but implicit for $\|$. Although we chose not to let $|||$ be available in constructing the programs of mobile codes, we can do so easily when needed.
- Finally, Rules 15 and 16 describe the semantics of executing a loop program *while c do p*: if c is evaluated to be true, then *while c do p* is reduced to p; *while c do p*: otherwise, the execution of the program terminates successfully, and is deleted from the resulting execution. This semantics is closely related to the one for the *while statement* in an imperative programming language.

3.3. Expressiveness of SRAL

In this section, we study the expressiveness of SRAL in terms of the trace model. Imagine we are observing the

execution of a particular program by a mobile code, and we record the shared resource accesses that are performed by the mobile code and the order in which this set of accesses are performed. When the execution terminates, we get a sequence of access operations, called a *trace* of the program. It represents the history information of a mobile code accessing shared resources. Given a program p of a mobile code, it is natural to model p by $traces(p)$—the set of all traces that program p can perform. We call $traces(p)$ the *trace model* of p. For example, $traces((a_1; a_2)) = \{\langle a_1, a_2\rangle\}$.

Given two traces t and u, we define the following operators:

- *Concatenation:* the concatenation of t and u is the trace in which t is followed by u and it is denoted by $t \hat{} u$.
- *Interleaving:* the interleaving of t and u is the trace model that results from all possible interleavings of t and u. It is denoted by $t \bowtie u$ and defined recursively as follows:
  - $t \bowtie \langle\rangle = \{t\}$.
  - $\langle\rangle \bowtie u = \{u\}$.
  - If neither t nor u is an empty trace, then $t \bowtie u = \{\text{head}(t) \hat{} y \mid y \in \text{tail}(t) \bowtie u\} \cup \{\text{head}(u) \hat{} y \mid y \in \text{tail}(u) \bowtie t\}$ where $\text{head}(t)$ is the first access action of t and $\text{tail}(t)$ is the trace consisting of the rest of the accesses.
- *Kleene closure:* the Kleene closure of t is the trace model consisting of zero or more concatenations of u. It is denoted by t^* and is defined as follows:
  - $\langle\rangle \in t^*$.
  - If $q \in t^*$, then $t \hat{} q \in t^*$.

We extend these operators to trace models. Given two trace models T and U, we define the following operators:

- *Union of T and U:* $T \cup U = \{u \mid u \in T \text{ or } u \in V\}$.
- *Concatenation of T and U:* $T \hat{} U = \{t \hat{} u \mid t \in T \text{ and } u \in U\}$.
- *Kleene closure:* the Kleene closure of T is denoted by T^* and is defined as follows:
  - $\langle\rangle \subseteq T^*$.
  - If $t \in T$ and $q \in T^*$, then $t \hat{} q \in T^*$.

By treating a trace model as a set, we can easily extend these operators to trace models. In summary, we define a set of rules needed to calculate the trace models for all possible SRAL programs.

Definition 3.2 (*Trace Model of Accesses*). The trace model of a mobile code program p can be characterized by the following rules, where p_1 and p_2 are the subprograms.

- $\text{traces}(a) = \{\langle a\rangle\}$ where a is a shared resource access.
- $\text{traces}(p_1; p_2) = \text{traces}(p_1) \hat{} \text{traces}(p_2)$.
- $\text{traces}(\textit{if } c \textit{ then } p_1 \textit{ else } p_2) = \text{traces}(p_1) \cup \text{traces}(p_2)$.
- $\text{traces}(p_1 \parallel p_2) = \text{traces}(p_1) \bowtie \text{traces}(p_2)$.
- $\text{traces}(\textit{while } c \textit{ do } p) = traces(p)^*$. □

Definition 3.3 (*Regular Trace Model*). We define *regular trace models* over a set of accesses A as follows:

- $\{a\}$ is a regular trace model where $a \in A$.
- If p_1 and p_2 are regular trace models, then $p_1 \cup p_2$, $p_1 \hat{} p_2$, p_1^* are regular trace models.
- A regular trace model can be obtained only by applying the above two rules a finite number of times. □

Theorem 3.4 shows that any regular trace model can be expressed by a SRAL program.

Theorem 3.4 (*Regular Completeness*). *For each regular trace model m over a set of accesses A, there exists a program P such that $traces(P) = m$.*

Proof. We prove it by induction.

1. *Induction basis:* if $m = \{\langle a\rangle\}$ where $a \in A$, then $traces(a) = m$.
2. *Induction hypothesis:* assume two regular trace models T and U, for which there exist programs P_T and P_U such that $traces(P_T) = T$ and $traces(P_U) = U$.
3. *Induction step:* We have
   - $traces(\textit{if } c \textit{ then } P_T \textit{ else } P_U) = T \cup U$ for some condition c.
   - $traces(P_T; P_U) = T \hat{} U$.
   - $traces(\textit{while } c \textit{ do } P_T) = T^*$.

   Therefore, given an arbitrary regular trace model m over A, there exists a mobile code program P such that $traces(P) = m$. □

Although SRAL is expressive enough for most applications, there exist some traces which cannot be specified by SRAL. For example, the trace model of resource r_1 being accessed n times followed by resource r_2 being accessed n times is not a regular trace model, and cannot be specified by SRAL. However, in practice, this can be achieved in an ad hoc fashion based on the underlying language which is usually Turing-complete.

3.4. Spatial constraints and constraint satisfaction checking

One important aspect of mobile execution is the spatial constraints concerning the order of shared resource accesses, and the satisfaction checking problem of a mobile code's program associated with these constraints. In this section, we propose a shared resource access constraint language, *SRAC*, which is based on the workflow temporal constraint language CONSTR [10]. Spatial constraints are defined over a mobile code's access actions. The constraint specification language must be expressive enough so that most security requirements related to synchronized accesses to shared resources can be represented. Meanwhile, the language must be simple and intuitive so that constraints can be reasoned about easily. A permission grant requires checking constraint satisfaction conditions at run-time right after a mobile code is authenticated and its role is activated. In the following, we define *SRAC* and show that the spatial access satisfaction checking problem is decidable.

Definition 3.5 (*Shared Resource Access Constraint Language*). We define a formula of the shared resource access constraint language *SRAC* to have the following form:

$$C ::= T \mid F \mid a \mid a_1 \otimes a_2 \mid \#(m, n, A) \mid C_1 \wedge C_2 \mid C_1 \vee C_2 \mid \neg C$$

We also define the implication connective as $C_1 \rightarrow C_2 ::= \neg C_1 \vee C_2$. □

Example 3.6. The following examples illustrate some possible constraint expressions and their meanings:

- a: access a must be performed by the mobile code.
- $a_1 \wedge a_2$: both a_1 and a_2 must be performed (in any order).
- $a_1 \vee a_2$: at least one of a_1 and a_2 is performed.
- $a_1 \otimes a_2$: one must first perform a_1 and then perform a_2 (possibly making other resource accesses in between).
- $a_1 \rightarrow a_2$: if a_1 is performed, then a_2 must be performed (either before or after a_1).
- $\#(0, 5, \sigma_{r=\mathrm{RSW}}(A))$: a restricted software (RSW) package, either a licensed or trial version, cannot be accessed by more than 5 times, no matter where the mobile code is run. σ is a selection operation over set A and returns a subset of accesses that meet certain conditions.

To establish the notion of shared resource access satisfaction, we first introduce the notions of trace satisfaction and trace model satisfaction.

Definition 3.7 (*Trace Satisfaction*). We define the satisfaction relationship $\models$ between a trace t and a constraint expression C as follows by structural induction on C. If $t \models C$ does not hold, we denote it by $t \not\models C$.

- $t \models T$ and $t \not\models F$.
- For $a \in A$, $t \models a$ if and only if $a \in t$ and $Pr_x(a) = true$, i.e. the access a is stated by the mobile code program and a has been performed before as indicated by an execution proof.
- For $a_1, a_2 \in A$, $t \models a_1 \otimes a_2$ if and only if there exist traces t_1 and t_2 such that $t_1 \hat{} t_2 = t$, $t_1 \models a_1$ and $Pr_x(a_1) = true$ and $t_2 \models a_2$ and $Pr_x(a_2) = true$.
- $t \models \#(m, n, A)$ if and only if $m \leq \#(A) \leq n$, where operator # gives the cardinality of a set.
- $t \models C_1 \wedge C_2$ if and only if $t \models C_1$ and $t \models C_2$.
- $t \models C_1 \vee C_2$ if and only if $t \models C_1$ or $t \models C_2$.
- $t \models \neg C$ if and only if $t \not\models C$. □

We extend the notion of satisfaction to one between mobile code programs and constraint expressions by using the shared resource access trace model.

Definition 3.8 (*Mobile Code Execution Satisfaction*). Given a mobile code's program P and a constraint expression C, we say P *satisfies* C (denoted by $P \models C$) iff $traces(P) \models C$. □

Note that given a program P, set $traces(P)$ might be infinite, e.g. P contains a loop construct, and to check if P satisfies a constraint expression C, we have to check for each $t \in traces(P)$ if $t \models C$. This seems to be undecidable when $traces(P)$ is infinite. The following theorem shows that the mobile code execution satisfaction checking can be performed in polynomial time.

Theorem 3.9 (*Mobile Code Execution Satisfaction Checking*). *Given a mobile code's program P specified in SRAL and a constraint expression specified in SRAC, the time complexity of checking if $P \models C$ is $O(m * n)$, where m is the size of P and n is the size of C.*

Proof. It can be proved by structural induction on P and C based on the following rules:

- $\mathbf{C} = \mathbf{T}$: $P \models T$.
- $\mathbf{C} = \mathbf{F}$: $P \not\models F$.
- $\mathbf{C} = \mathbf{a}$:
  - $P \not\models a$ for $P = \alpha?v, \alpha!a, signal(\xi)$, or $wait(\xi)$.
  - $a' \models a$ ($a = a'$ *and* $Pr_x(a) = true$), otherwise $a' \not\models a$.
  - $p_1; p_2 \models a$ iff $p_1 \models a$ or $p_2 \models a$.
  - *if* c *then* p_1 *else* $p_2 \models a$ iff $p_1 \models a$ and $p_2 \models a$.
  - $p_1 \parallel p_2 \models a$ iff $p_1 \models a$ or $p_2 \models a$.
  - *while* c *do* $p \models a$ iff $p \models a$.
- $\mathbf{C} = a_1 \otimes a_2$:
  - $P \not\models a_1 \otimes a_2$ for $P = a, \alpha?X, \alpha!a, signal(\xi)$, or $wait(\xi)$.
  - $p_1; p_2 \models a_1 \otimes a_2$ iff (1) $p_1 \models a_1 \otimes a_2$, or (2) $p_2 \models a_1 \otimes a_2$, or (3) $p_1 \models a_1$ and $p_2 \models a_2$.
  - *if* c *then* p_1 *else* $p_2 \models a_1 \otimes a_2$ iff $p_1 \models a_1 \otimes a_2$ and $p_2 \models a_1 \otimes a_2$.
  - $p_1 \parallel p_2 \models a_1 \otimes a_2$ iff $p_1 \models a_1 \otimes a_2$ or $p_2 \models a_1 \otimes a_2$.
  - *while* c *do* $p \models a_1 \otimes a_2$ iff $p \models a_1 \otimes a_2$.
- $\mathbf{C} = C_1 \wedge C_2$: $P \models C$ iff $P \models C_1$ and $P \models C_2$.
- $\mathbf{C} = C_1 \vee C_2$: $P \models C$ iff $P \models C_1$ or $P \models C_2$.
- $\mathbf{C} = \neg C'$: $P \models C$ iff $P \not\models C'$. □

3.5. Spatial constraints enforcement

The role-based access control (RBAC) [14] model is widely-used in system security. It consists of four basic components: a set of users (*User*), a set of roles (*Role*), a set of permissions (*Permission*) and a set of subjects (*Subject*). A user is a human being, e.g. the security officer, or a mobile code. A role represents a collection of permissions needed to perform a certain job function, and a permission is an access operation that can be exercised on objects in the system. A subject relates a user to possibly many roles. When a user logs in the system after authentication, he establishes some subject(s), by which he can request activation of some of the roles he is authorized to perform. A role becomes active only if it is enabled at the time of the request and the user requesting its activation is entitled to activate it at that moment. After the activation of a role, the corresponding user obtains the permissions associated with that role under the restrictions of related constraints.

In the previous sections, we have defined the SRAC language to express the spatial constraints forshared resource access control in mobile execution. We extend the RBAC model to incorporate SRAC as its constraint definition language. The Security Officer of a system defines the specific constraints to meet the spatial requirements for a coalition environment. To enforce these constraints, we need to introduce a new type of state to permissions: a permission is *active* iff its associated role is assigned to the subject in a session and the related spatial constraints are satisfied. That is

$$\begin{aligned}&(\forall perm \in Permission) active(perm) = true \Leftrightarrow \\ &(\exists r \in Role, \exists s \in Subject) r \in AR(s) \wedge perm \in RP(r) \wedge \\ &check(P, C) = true\end{aligned} \tag{1}$$

where $AR(\cdot)$ and $RP(\cdot)$ are functions, which map from a subject to a set of its active roles and from a role to its assigned

permissions respectively. $Check(P, C)$ determines whether the mobile code's program P satisfies the constraint C associated with permission *perm*. This procedure is carried out by a *spatial constraint checking module* which applies the approach in Section 3.4 to analyze the traces and execution proofs of a mobile code. So, when a mobile code does not meet the constraints of a permission for shared resource accesses in a coalition environment, that permission will not be active and is not granted consequently.

4. Temporal constraints

Timing constraints are essential for controlling time-sensitive activities in various applications. For example, in a workflow management system tasks usually have timing constraints and need to be executed in a certain order. The existing timing constraints in TRABC [6] and GTRABC [22] are based on intervals of periodic events that are imposed on roles. Each interval uses the discrete time model with explicit beginning and ending time points, indicating the time when a role becomes enabled or disabled. Since the periodic constraints are imposed on roles in TRBAC, a disabling event of a role would revoke all of its granted privileges. Considering the fact that different permissions authorized to a role often have different temporal constraints, more roles need to be defined in TRBAC. Moreover, because there is no global clock in distributed systems and the arrival time of a mobile code on a host is unpredictable, interval timing models are not appropriate to characterize the time-sensitive activities of mobile execution on different hosts. To tackle these problems, we introduce a continuous time model in defining the temporal constraints, and use time durations.[4] Instead of associating periodic events with roles, we change the states of permissions according to the result of constraint checking so as to avoid complicated permission management.

We assume a continuous model of time, which is isomorphic to the set of real numbers ($\mathbb{R}$) with a total order relation $<$. The entire life-cycle of a mobile code, i.e. its overall execution time spread over multiple hosts in a coalition environment, constitute a time line for us to describe its behaviors by using temporal logic [12].

To meet the temporal requirements in accessing time-sensitive resources, each permission is associated with a *validity duration*, which specifies the length of the time period when the permission can be granted to a subject. The validity duration can be any positive real number and even infinity which means the corresponding resource is time-insensitive. With respect to the validity of time duration, each permission can be in one of three different states for a mobile code: *inactive*, *active-but-invalid*, and *valid*. A permission is *inactive* if it is not assigned to any role of the requesting subject or it is assigned to a role of the subject but that role has not been activated in a session. An active permission becomes invalid when the validity duration of the permission expires.

We model the states of permissions by boolean-valued functions over time: $\mathbb{T}\text{ime} \rightarrow \{0, 1\}$ where $\mathbb{T}\text{ime}$ is the set of the real numbers. The function reflecting whether a permission of a role is valid at certain time point is: $valid_r \in Permission \times \mathbb{T}\text{ime} \rightarrow \{0, 1\}$, where $valid_r(perm, t) = 1$ means that the permission *perm* of role r is valid at time t, and $valid_r(perm, t) = 0$ means that permission is in an invalid state at time t. Then, the duration of a permission in the valid state over a time interval is the accumulated time in which the state is present in the interval.

Let $[b, e]$ be an interval, i.e. $b, e \in \mathbb{T}\text{ime}$ and $e \geq b$. The duration of state *valid* for a permission *perm* of a role r over $[b, e]$ equals the integral

$$\int_b^e valid_r(perm, t)\mathrm{d}t.$$

The temporal constraints specify that the duration when a permission remains in the valid state must be no more than that permission's validity duration, i.e.

$$(\forall perm \in Permission)(\forall t \in \mathbb{R})\ valid_r(perm, t) = 1 \Leftrightarrow active_r(perm, t) = 1 \wedge \int_{t_b}^{t} valid_r(perm, u)\mathrm{d}u \leq dur(perm) \quad (2)$$

where $dur(\cdot)$ returns the specified validity duration associated with a permission, $active_r(perm, t)$ is a variant of Expression (1) by using a boolean-valued function which is similar to $valid_r(perm, t)$. We can apply two control schemes by assigning different values to the base time t_b. Suppose a mobile code has visited hosts $h_1, \ldots, h_{i-1}$ in that order before roaming to host h_i. If $t_b = t_i$ which denotes the arrival time of the mobile code at the current host h_i, then the temporal constraint restricts the validity of a permission only on the host h_i. On the other hand, if $t_b = t_1$ which is the arrival time at the first host h_1, i.e. the starting time of execution by the mobile code, then the temporal constraint is applicable to the mobile code's entire execution on different hosts.

According to Theorem 3.9 and the decidability theorem of Duration Calculus [21], we obtain the following theorem.

Theorem 4.1 (*Permission Validity Checking*). *Let constraint* $C ::= active_r(perm, t) = 1 \wedge \int_{t_b}^{t} valid_r(perm, u)\mathrm{d}u \leq dur(perm)$ *for the validity of a permission perm at time* t. $P, [t_b, t] \models C$ *iff Expression (2) is satisfied. Then given a mobile code program* P *specified in SRAL, a constraint expression specified in SRAC and a time interval* $[t_b, t]$*, the problem of checking if* $P, [t_b, t] \models C$ *is decidable.*

This *permission validity checking* was implemented as a module in our coordinated access control system. It receives the specification of a mobile code's program P, the time interval $[t_b, t]$, and the index of a permission in question, as inputs. Then it calls the *spatial constraint checking module* described in Section 3.5 to determine whether the related constraints are satisfied. The temporal constraints will be checked by comparing the integral of the valid state function with the

[4] An interval is defined by its explicit starting and ending time points. Duration defines the span of a period of time, without specifying the starting time point.

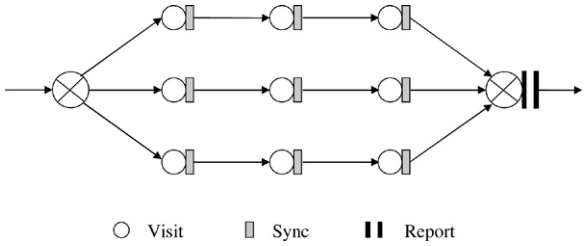

Fig. 2. An example of parallel teamwork.

validity duration of the permission. A boolean result is returned as an indication of whether the spatio-temporal constraints are satisfied or not. If not satisfied, an *event* will be triggered to set the boolean-valued function $valid_r(\cdot,\cdot)$ to 0, so that the permission is not granted to the mobile code.

5. Implementation

Based on the above formal framework, a coordinated access control prototype has been implemented in the Naplet mobile agent system [33]. The Naplet system is an experimental framework in support of Java-compliant mobile agents. It provides constructs for agent declaration, confined agent execution environments, and mechanisms for agent monitoring, control, and communication. It features a structured navigation facility, reliable agent communication mechanism, proportional-shared resource management strategies, and secure interface between agents and naplet servers. We use the Naplet system to emulate the behaviors of mobile users and migrating execution units in computational grids. A extended role-based access control model has been realized in Naplet and it provides the specification and enforcement mechanisms for spatio-temporal constraints.

Naplet-based mobile distributed systems are built upon a first-class `Naplet` object. It is an abstract of agents, defining hooks for application-specific functions to be performed in different stages of its life cycle in each server and an itinerary for its way of traveling among naplet servers. We can use the agent itinerary to describe the roaming agenda of a mobile entity, i.e. the list of hosts to be visited and their ordering. Fig. 2 shows a parallel teamwork in mobile execution, in which three mobile entities visit hosts concurrently and each of them visits a group of hosts in sequence. At the end of each visit, mobile entities need to communicate with each other about their latest results. At the end of the execution, partial results from the three mobile entities will be merged to find the final result. This mobile execution example can be emulated easily in the Naplet system, where a parent agent creates three cloned agents to carry out the same tasks of the mobile entities. Each cloned agent visits the naplet servers according to its sequential itinerary and access server resources in the execution of its code. Synchronization among the cloned agent is realized by the inter-agent communication, and their results will be reported to the parent agent at the end of their lifetime.

5.1. Role-based access control in naplet

To realize role based access control in the Naplet system, we defined an agent as an authenticated subject representing the source of a request by taking advantage of the subject-based security features in Java. It is comprised of a set of principals, based on which access control is implemented.

We defined a set of additional permission types to control the access rights of naplet subjects over restricted services on naplet servers.

- *NapletRuntimePermission(target)*
  This is a subclass of java.lang.BasicPermission, and it offers a simple naming conversion. For example, NapletRuntimePermission("land") denotes the permission for landing agents and NapletRuntime Permission("clone") for cloning agents.
- *NapletServicePermission(service, actions)*
  This permission class represents a permission of access to the target service. This can be granted either to naplet agents or to the local server administrator to manage the system.
- *NapletSocketPermission(server, actions)*
  This permission class represents permissions granted to the naplet subject to communicate with a server. The target server in the Naplet system can be represented as a string in the format of "hostname:port/naplet-server-name".

Role is an important component in the role-based access control model. It associates permissions to users and facilitates the permission management. The *Principals* in Java provide a mechanism to gather permissions for a job function. We defined roles and specified security policies based on Principals in the Naplet system. Currently, there are three types of Principals:

- *NapletPrincipal*
  This Principal class defines the roles of different naplet agents. Naplet is a first class subject that acts on behalf of the agent owner. The naming scheme of NapletPrincipal is in the format "naplet-owner/naplet-name:naplet-version".
- *NapletOwnerPrincipal*
  This Principal class defines the roles of the agents' owners. A naplet owner can create, dispatch and terminate her agents. We adopted a Kerberos-like naming scheme in the format "user-account-name\hostname".
- *NapletServerAdministrator*
  This Principal class defines the role of a server administrator. Only the administrator has the privileges to setup and manage naplet servers, including specifying the access control policy. We used the "user-account-name" for its naming.

On arrival at a server, a naplet agent must be authenticated based on the certificate of its owner issued by an authority or via a priori registration. Upon the success of authentication, a naplet subject will be created and it has a principal of class `NapletPrincipal`. This naplet agent will then be able to perform operations allowed by the `NapletPrincipal` role. That is the naplet server delegates the naplet execution to the subject of the naplet itself. For example, when a "HelloNaplet" arrives at a NapletServer:

Subject napSubject = Runtime.authenticate(helloNaplet);
AccessController.checkPermission();

```
Subject.doAsPrivileged(napSubject,
   new PrivilegedAction() {
        helloNaplet.init(); helloNaplet.onStart(); }, null );
```

This process realizes role activation in the Naplet system. The role-permission assignment is achieved by defining the Java policy files. The `grant` statements associate the permissions to principals. For example, in a policy file we specify the following permission grant:

```
grant Principal NapletPrincipal "song\wayne.edu/MyNaplet" {
   permission NapletRuntimePermission("land");
   permission NapletServicePermission( "yellow-page",
        "lookup");
   permission NapletSocketPermission(
        "ece.wayne.edu:2400/YellowPage", "talk, travel");
}
```

which authorizes the "MyNaplet" to visit the naplet server, look up a yellow page service, talk to naplets running on a remote server and migrate to that remote server.

5.2. Coordinated access control in naplet

To realize coordinated access control in Naplet, we extended the above role-based access control system by incorporating a construction of the agent programs and enforcing the spatio-temporal constraints.

The SRAL prototype has been implemented in recursively constructed resource access patterns. Its base is a `Singleton` pattern, comprising of a single shared resource access at a server guarded by a *pre-condition*. Over the set of access patterns, we define three composite operators: `SeqPattern` and `ParPattern`, and `Loop`, as defined in SRAL to recursively form resource accesses of regular trace models. The `SeqPattern` and `ParPattern` constructs implement a sequential accessing pattern $(p_1; p_2)$ and a concurrent accessing pattern $(p_1 \parallel p_2)$, respectively. The `Loop` construct defines repetition of an access program until a pre-condition no longer holds.

To show the programmability of the constructs, we give an example of access patterns recursively constructed from accesses and conditional accesses. Consider a mobile application over n servers $s_1, \ldots, s_n$. The following class `ApplAgentProg` defines a parallel execution pattern by the use of k cloned naplet agents, each for an equal share of the servers (for simplicy, we assume n can be divided by k). Each naplet runs a guardian function defined in the `Checkable` object `ResultVerify` before each access. The naplets report their results to home at the end of their execution. Note that the class `ApplAgentProg` is declared as a private inner class of the naplet so that the access pre-condition can be defined based on the naplet states.

```
private class ApplAgentProg extends Access {
   private ApplAgentProg(String[] accesslist, int k) {
      Checkable guard = new ResultVerify();
      Operable report = new ResultReport();
      int n = accesslist.length; int m = n/k;
      AccessPattn[][] sng = new AccessPattn[k][m];
      AccessPattn[] seqProg = new AccessPattn[m];
      for (int i=0; i < k; i++) {
         for (int j=0; j < m; j++)
            sng[i][j]=new AccessPattn(guard,accesslist[i*k+j]);
         AccessPattn[i]=new SeqPattn(sng[i],report);
      }
      setProgram(new ParPattn(seqProg));
   }
}
```

When a naplet server authorizes a visiting naplet to perform a shared resource access according to the naplet's program, an execution proof is added to a *proof log*, which is similar to the append-only log in Ajanta [23]. The spatio-temporal constraint checking modules use the execution proofs in this log to verify whether the agent program satisfies the constraint. We utilized the `checkPermission()` method of the JRE to enforce coordinated access control policies when a resource access request is received. The Naplet system defines a NapletSecurityManager for customized permission checks. For example, the following code checks the permissions for looking up the yellow page service.

```
public class NapletSecurityManager extends SecurityManager {
  public void checkPermission( Permission perm) {
   if (perm instanceof NapletServicePermission) {
      AccessControlContext acc = AccessController.getContext();
      Subject sbj = Subject.getSubject(acc);
      String p_serv = perm.getName();
      String p_acts = perm.getActions();
      if (sbj.getPrincipals() contains "song\wayne.edu/MyNaplet"
        && p_serv == "yellow-page" && p_acts contains acts)
         if (NapletAccessControl.spatialConsCheck(sbj, acts)
           && NapletAccessControl.temporalConsCheck(sbj, acts)) {
            super.checkPermission( perm );
            return;
         }
   throw new SecurityException("MyNaplet from song is denied");
   }
}
```

6. Related work

The number of proposals for grid security acknowledges the importance of providing grid systems with authentication, confidentiality, integrity, and access control, to mention just a few security aspects. This section reviews the relevant work on access control and discusses their limitations when considering temporal and spatial security requirements in accessing grid resources.

GSI (Grid Security Infrastructure) [16] is a security architecture supplied by the Globus Toolkit [15]. It is composed

of libraries and tools that provide the security functionality needed for a grid application. A key component of GSI is the Community Authorization Service (GAS), which allows a virtual organization to express policy regarding resources distributed across a number of sites.

VOMS (Virtual Organization Membership Service) [2] is a system for managing authorization data within multi-institutional collaborations. It provides a database of user roles and capabilities, as well as a set of tools for accessing and manipulating the database. The database contents are used to generate grid credentials for users when needed. The model to specify permissions is quite expressive, allowing detailed policy definitions.

Sesame [35] is a dynamic, context-aware access control mechanism for pervasive grid applications. It specifies DRBAC, an extended RBAC-based access control mechanism that dynamically adjusts role assignments to users based on their context information. Sesame access control decisions, in contrast to CAS and VOMS, are not based solely on identity information, but also on additional information such as network conditions.

The preceding security infrastructures allow system administrators to define access control policies to protect grid resources. However, they do not take the temporal and spatial security requirements into account and provide no support to express such access constraints for grid-based mobile execution.

There were recent temporal access control models for monitoring time-sensitive activities. For example, Bertino et al. [5] presented a time-based scheme by using periodic authorizations and derivation rules. Each authorization is assigned a periodicity constraint which specifies a set of intervals when the authorization is enabled. Later on, the authors integrated such interval-based temporal constraints into role-based access control and developed a temporal RBAC (TRBAC) model to efficiently manage permissions of temporal roles [6]. TRBAC was recently generalized by Joshi et al. [22] by incorporating a set of language constructs for the specification of various temporal constraints on roles, user-role assignments and role-permission assignments. TRBAC models can be tailored to support temporal constraints in mobile execution, but they can't accommodate spatial constraints because they don't consider the relevance of authorization and performed access actions. In addition, the discrete time structure in these access control models affects their applicability to real systems, because in most cases time is continuous.

History-based access control may be viewed as a pragmatic approximation to information-flow control that keeps track of code execution. In [1], the run-time rights of a piece of code are determined systematically by examining its execution history, i.e. the attributes of the code that have run before and any explicit requests to augment rights. But this mechanism only inspects the execution history on the local site. As a result, it cannot be applied to access control in a coalition environment, where the authorization decision depends on the access actions on other related sites. Little work has considered the spatial requirements for mobile computing security. In [11], Guy et al. presented a mechanism to use a selective history of the access requests of a mobile code in access control. Their history-based access control model is limited to one-hop mobile codes. Because it has no concept of shared resource (objects), this model is hardly extensible to include the access history of a mobile computation in a coalition environment.

Mobile code provides a special model for mobile execution in grids, where autonomous mobile codes travel across networks carrying out their tasks on behalf of their owners. There is much work focusing on the security issues of mobile code systems (see [18] for a comprehensive review). Among them, protecting the docking service from attacks by malicious or malfunctioning mobile codes is important for service providers. A most noteworthy approach is the proof-carrying-code (PCC) [28], which allows a host to determine with certainty if it is safe to execute a remote code. To safely grant the privileges to a mobile code, Farmer et al. [13] proposed a state appraisal approach to ensure that a mobile code with corrupted states won't be granted any privilege. In [34], we prototyped a mobile code oriented authorization mechanism that realized privilege delegation and access control. However, these approaches do not consider the spatio-temporal requirements for host protection in mobile execution, either. Safety of a mobile agent itinerary was analyzed in [25]. A formal framework was proposed to specify an agent itinerary and reason its safety with regard to whether the visit trace of the agent satisfies its itinerary specification.

Access control in coalition systems has been investigated in many studies. For example, Cohen et al. [9] proposed an access control model that identifies coalition entities and their interrelationships. Their focus is on how to construct a coalition system by imposing constraints on relationships between coalition entities. Freudenthal et al. [17] proposed a distributed role-based access control mechanism (dRBAC) in coalition environment. In essence, dRBAC is a trust manager that defines, delegates and monitors trust relations of coalition entities. In this paper, we investigate the temporal and spatial security requirements on accessing shared resources by mobile devices in a coalition environment. We concentrate on modeling access actions of mobile devices, proposing and enforcing spatio-temporal constraints on these actions. Our coordinated access control mechanism facilitates monitoring and controlling time-sensitive actions to some shared resources on one host or on multiple hosts that have a spatial coalition.

7. Conclusions

In this paper, we present a coordinated access control model with temporal and spatial constraints for secure mobile execution in grids. The various access patterns of mobile entities are expressed by the expressive SRAL language. The constraint satisfaction problem is decidable. To prove concept and technical feasibility of our coordinated access control model, we implemented it in the Naplet mobile agent system, which emulates grid-based mobile execution by software agents. To monitor and control execution of mobile

entities, we need to record traces of their critical action and transfer these traces among hosts. This incurs communication overhead among hosts. In addition, generating consistent action traces in a secure way is challenging, especially in a coalition environment across multiple administrative domains. The security managers of coalition hosts need to be modified by adding code that enforces spatio-temporal constraints on accesses by the mobile entities. The specification of a security policy with temporal and spatial constraints is included in a text file protected by cryptography.

To extend our work, we will look into permission inheritance in role hierarchy imposed by the spatio-temporal constraints. Based on the emulation results in the Naplet system, we will investigate the impact on performance of mobile execution by enforcing our security constraints and design an optimal migration scheme for a mobile entity in grids to maximize performance. We will compare the execution time of an access operation with and without enforcing our spatio-temporal constraints, and derive the operation overhead of access control in one host and in the entire coalition system. We will try to construct an optimal set of constraints that satisfies the temporal and spatial access control requirements and such that its size is minimal. This will reduce the overhead of constraint satisfaction checking at runtime. We will also investigate the privilege delegation among mobile entities and trust management in our coordinated access control system.

References

[1] M. Abadi, C. Fournet, Access control based on execution history, in: Proc. of the 10th Annual Network and Distributed System Symp., NDSS, February 2003.

[2] R. Alfieri, R. Cecchini, V. Ciaschini, et al., Voms: An authorization system for virtual organizations, in: Proc. of the 1st European Across Grids Conf., 2003.

[3] Argonne National Laboratory. A Grid-Enabled Service for High-Throughput Genome Analysis, 2005. http://compbio.mcs.anl.gov/Papers/GADU_GGF.pdf.

[4] F. Berman, et al., Adaptive computing on the grid using AppLeS, IEEE Transactions on Parallel and Distributed Systems 14 (4) (2003) 369–382.

[5] E. Bertino, C. Bettini, E. Ferrari, P. Samarati, An access control model supporting periodicity constraints and temporal reasoning, ACM Transactions on Database Systems 23 (3) (1998) 231–285.

[6] E. Bertino, P.A. Bonatti, E. Ferrari, TRBAC: A temporal role-based access control model, ACM Transactions on Information and System Security 4 (3) (2001) 191–233.

[7] D.C. Chu, M. Humphrey, Mobile OGSI.NET: Grid computing on mobile devices, in: Proc. of the 5th IEEE/ACM Intl. Workshop on Grid Computing, GRID'04, 2004.

[8] C. Clark, K. Fraser, S. Hand, et al., Live migration of virtual machines, in: Proc. of USENIX Networked Systems Design and Implementation, NSDI, 2005.

[9] E. Cohen, R.K. Thomas, W. Winsborough, D. Shands, Models for coalition-based access control (CBAC), in: Proc. of the 7th ACM Symp. on Access Control Models and Technologies, SACMAT, 2002.

[10] H. Davulcu, M. Kifer, C. Ramakrishnan, I. Ramakrishnan, Logic based modeling and analysis of workflows, in: Proc. of the ACM Symp. on Principles of Database Systems, Seattle, WA, USA, June 1998, pp. 25–33.

[11] G. Edjlali, A. Acharya, V. Chaudhary, History-based access control for mobile code, in: Proc. of ACM Conf. on Computer and Communications Security, CCS, 1998, pp. 38–48.

[12] E.A. Emerson, Temporal and modal logic, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Elsevier Science Publishers B.V., 1990, pp. 997–1067.

[13] W. Farmer, J. Guttman, V. Swarup, Security for mobile agents: Authentication and state appraisal, in: Proc. of the 4th European Symp. on Research in Computer Security, ESORICS'96, September 1996, pp. 118–130.

[14] D.F. Ferraiolo, J.F. Barkley, D.R. Kuhn, A role based access control model and reference implementation within a corporate intranet, ACM Transactions on Information and System Security 2 (1) (1999) 34–64.

[15] I. Foster, C. Kesselman, Globus: A metacomputing infrastructure toolkit. http://www.globus.org.

[16] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke, A security architecture for computational grids, in: Proc. of the ACM Conf. on Computer and Communications Security, CCS, 1998.

[17] E. Freudenthal, T. Pesin, L. Port, E. Keenan, V. Karamcheti, dRBAC: Distributed role-based access control for dynamic coalition environments, in: Proc. of the 22nd Intl. Conf. on Distributed Computing Systems, ICDCS, 2002.

[18] S. Fu, C.-Z. Xu, Mobile code and security, in: Handbook of Information Security, John Wiley & Sons, 2004.

[19] S. Fu, C.-Z. Xu, Service migration in distributed virtual machines for adaptive grid computing, in: Proc. of the 34th Intl. Conf. on Parallel Processing, ICPP, 2005.

[20] A. Fuggetta, G. Picco, G. Vigna, Understanding code mobility, IEEE Transaction on Software Engineering 24 (5) (1998) 352–361.

[21] M.R. Hansen, C. Zhou, Duration calculus: Logical foundations, Formal Aspects of Computing 9 (1997) 283–330.

[22] J.B.D. Joshi, E. Bertino, A. Ghafoor, Temporal hierarchies and inheritance semantics for GTRBAC, in: Proc. the 7th ACM Symp. on Access Control Models and Technologies, SACMAT'02, Monterey, CA, June 2002, pp. 74–83.

[23] N.M. Karnik, A.R. Tripathi, A security architecture for mobile agents in Ajanta, in: Proc. of the 20th Intl. Conf. on Distributed Computing Systems, ICDCS, 2000.

[24] D. Kondo, M. Taufer, C. Brooks, H. Casanova, A. Chien, Characterizing and evaluating Desktop Grids: An empirical study, in: Proc. of the 18th IEEE Intl. Parallel and Distributed Processing Symp., IPDPS, 2004.

[25] S. Lu, C. Zhong Xu, A formal framework for agent itinerary specification, security reasoning and logic analysis, in: Proc. of ICDCS Workshop on Mobile Distributed Computing, MDC, 2005.

[26] D. Milojicic, F. Douglis, Y. Panedeine, R. Wheeler, S. Zhou, Process migration, ACM Computing Surveys 32 (3) (2000) 241–299.

[27] National Center for Atmospheric Research (NCAR). Climate Change Research (CCR), 2004. http://www.cgd.ucar.edu/ccr/.

[28] G.C. Necula, Proof-carrying code, in: Conf. Record of POPL'97: The 24th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, Paris, France, January 1997, pp. 106–119.

[29] NEESgrid. Building the National Virtual Collaboratory for Earthquake Engineering, 2004. http://www.neesgrid.org/.

[30] S. Osman, D. Subhraveti, G. Su, J. Nieh, The design and implementation of Zap: A system for migrating computing environments, in: Proc. of the 5th USENIX Symp. on Operating Systems Design and Implementation, OSDI, 2002.

[31] T. Phan, L. Huang, C. Dulan, Challenge: Integrating mobile wireless devices into the computational grid, in: Proc. of the 8th ACM Intl. Conf. on Mobile Computing and Networking, MobiCom, 2002.

[32] C.P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M.S. Lam, M. Rosenblum, Optimizing the migration of virtual computers, in: Proc. of the 5th Symposium on Operating Systems Design and Implementation, OSDI, December 2002, pp. 377–390.

[33] C.-Z. Xu, Naplet: A flexible mobile agent framework for network-centric applications, in: Proc. of the 2nd Intl. Workshop on Internet Computing and E-Commerce (In conjunction with IPDPS'02), 2002.

[34] C.-Z. Xu, S. Fu, Privilege delegation and agent-oriented access control in naplet, in: Proc. of the Intl. Workshop on Mobile Distributed Computing (In conjunction with ICDCS'2003), June 2003.

[35] G. Zhang, M. Parashar, Dynamic context-aware access control for grid applications, in: Proc. of the 4th Intl. Workshop on Grid Computing, GRID, 2003.

Song Fu received the BS degree in computer science from Nanjing University of Aeronautics and Astronautics, China, in 1999, and the MS degree in computer science from Nanjing University, China, in 2002. He is currently a Ph.D. candidate in computer engineering at Wayne State University. His research interests include resource management, security, and mobility issues in wide-area distributed systems.

Cheng-Zhong Xu received the BS and MS degrees in computer science from Nanjing University in 1986 and 1989, respectively, and the Ph.D. degree in computer science from the University of Hong Kong in 1993. He is an Associate Professor in the Department of Electrical and Computer Engineering of Wayne State University. His research interests lie in distributed and parallel systems, particularly in resource management for high performance cluster and grid computingand scalable and secure Internet services. He has published more than 100 peer-reviewed articles in journals and conference proceedings in these areas. He is the author of the book *Scalable and Secure Internet Services and Architecture* (CRC Press, 2005) and a co-author of the book *Load Balancing in Parallel Computers: Theory and Practice* (Kluwer Academic, 1997). He serves on the editorial boards of J. of Parallel and Distributed Computing, J. of Parallel, Emergent, and Distributed Systems, J. of High Performance Computing and Networking, and J. of Computers and Applications. He was the founding program co-chair of the International Workshop on Security in Systems and Networks (SSN), the general co-chair of the IFIP 2006 International Conference on Embedded and Ubiquitous Computing (EUC'06), and a member of the program committees of numerous conferences. His research was supported in part by the US National Science Foundation, NASA, and Cray Research. He was a recipient of the Faculty Research Award of Wayne State University in 2000, the President's Award for Excellence in Teaching in 2002, and the Career Development Chair Award in 2003. He is a senior member of the IEEE.