Security Patterns: A Method for Constructing Secure and Efficient Inter-Company Coordination Systems

Nobukazu Yoshioka
National Institute of Informatics
nobukazu@nii.ac.jp

Shinichi Honiden
National Institute of Informatics and
The University of Tokyo
honiden@nii.ac.jp

Anthony Finkelstein
Dept. of Computer Science,
University College London
a.finkelstein@cs.ucl.ac.uk

Abstract

As the Internet, intranets and other wide-area open networks grow, novel techniques for building distributed systems, notably mobile agents, are attracting increasing attention. This is particularly the case for inter-company system coordination applications. A key difficulty in constructing such systems is to meet the security requirements while at the same time respecting the requirements for efficient implementation. In this paper, we propose a method that addresses this problem and show an application of the method to a real implemented system, the Environmentally Conscious Product (ECP) design support system. Our approach enables developers to specify several candidate system behaviors that satisfy the security requirements. We use patterns for this purpose. Patterns are abstract templates of system behavior fragments. The patterns include agent migrations, communications between applications and security procedures. We model the performance data associated with each pattern. Developers can then select an efficient implementation using this model to compare the performance data of the candidates. We evaluate our approach with a significant real-world example, the ECP design support system that essentially requires inter-company system coordination.

1 Introduction

Security is a very important consideration in developing inter-company coordination systems. For example, though companies may seek to collaborate in a supply chain framework, they also have data they wish to keep confidential from their collaborators. Security engineering is however very complex. In order to develop systems without security holes, a method that supports the engineer in identifying and analyzing the various issues is required. Such issues include who can access which hosts and which networks, and which data used in the system can be opened and by whom. In addition, because the performance of security measures is rather inefficient, unnecessary security measures should be avoided.

It is difficult even for experienced developers to design an efficient system that takes security measures into consideration. For example, in some cases, a design realized by remote messaging is more efficient when security issues are ignored, whereas a design that incorporates mobile agents may be more efficient if security is considered.

In this paper, we present a development method that tackles these problems and show an application of the method to a real implemented system, the Environmentally Conscious Product (ECP) design support system. Our approach enables developers to specify several candidate system behaviors that satisfy the requirements using patterns. Patterns are abstract templates of fragments of system behavior. They include agent migrations, communications between applications and security procedures. We model the performance data associated with each pattern. Developers can select an efficient implementation by comparing the performance data associated with each of the candidates.

This paper is organized as follows. Section 2 describes the motivations of our use of security patterns to construct an inter-company system. In Section 3, we set out our approach. In Section 4, we show part of the case study to which we apply our approach. We compare the time for messaging and security measures for different models which keep different security policies, and we evaluate our approach on that basis. In Section 5, we compare our approach with some existing techniques. Section 6 consists of concluding remarks.

2 Motivations

In this paper, we define inter-company coordination systems as integrated systems connecting heterogeneous ap-

plications and DBs in some companies. Such coordination systems have the following features.

1. Heterogeneous networks: Although the network bandwidth within each company may be broad, the networks connecting companies are narrow or limited. In other words, the speed of extra-nets is different than the speed of intra-nets. It implies that narrow-band networks may become bottlenecks when the data passing through the networks are big.
2. Security is essential: Although inter-company coordination systems are across some companies, the each function of the systems belonging to a particular company includes data concealed from other companies. It implies that security is essential, so that we must consider appropriate authentication and encryption.
3. Distributed management: Each organization administers each network and maintains applications belonging to the organization. From maintenance point of view, mobile agents are convenient because we can maintain application programs migrating to other organization from the original host.

We not only have to prevent security holes, but also to consider appropriate implementations to realize security requirements. Development of such systems has been impeded due to the difficulties described below. In this paper we propose a solution, namely, a methodology that overcomes these difficulties.

In this paper, we take up mobile agent technologies to develop coordination systems, because last two features indicate that the technology is convenient in term of both security and management. Thus, programs which we design and develop include not only remote procedure calls (RPCs) but also code migrations for local access. In other words, we model each coordination program as a mobile agent. In this paper, we only consider two protocol types: RPCs and migrations due to simplify the explanation. Therefore we must select access protocols for each application accessed by coordination programs. Our method includes the selection steps and guidelines of the selection.

One of the difficulties is that we cannot determine which implementation techniques are better for which parts of systems unless we consider various aspects. So, appropriate implementation details cannot be chosen without using guidelines or a methodology. To solve the problems, we prepare patterns, which are candidates of implementation, and consider the applying costs for guidelines of the selection.

In inter-company coordination systems, the security issue is unavoidable. However, implementation of excessive security facilities in systems should be avoided because of the adverse impact on performance. Therefore, in regard to security, appropriate implementation details may be different. For appropriate security implementation, we consider the security implementation in our patterns, extra time for security measures, and application contexts for the pattern.

We suppose that the application domain of our method is the followings:

- Applications gathering relative product information. The information is in DBs belonging to some companies.
- Applications retrieving data from some companies for simulations of designing products.
- SCM (Supply Chain Management) software which gets orders from customers, calculates production plans, and puts the plan to factories and subcontract companies.

The above applications include access to DBs, invocations of simulators and some calculations using these data. The amount of the data may be large when products consist of much numbers of parts.

We investigated a new development method for inter-company distributed systems to resolve these issues and applied it to a practical system. This method is explained in Section 3. An application example of the method is presented in Section 4.

3 OUR APPROACH

3.1 Overview

In this section, we set out our approach and address the problems described in Section 2. Our approach enables the developers to construct detailed specifications of the system behaviors that meet the system requirements and the existing resources such as hardware infrastructure and application software by applying patterns. Patterns are abstract templates of fragments of system behavior. They include agent migrations, communications between applications and security procedures.

Figure 1 gives on overview of the process by which patterns are applied. The proposed approach is as follows.

1. Construction of the abstract system specifications.

   Our approach draws on standard techniques from UML-based software development. We assume that the system requirements are given in three kinds of input models: **Abstract Sequence Charts**, **Agent Workflows**, and **Abstract Deployment Diagrams**. These specifications do not fully specify system behavior that indicates access protocols to applications or how security is to be implemented. The **Abstract Sequence Chart** describes the communication between

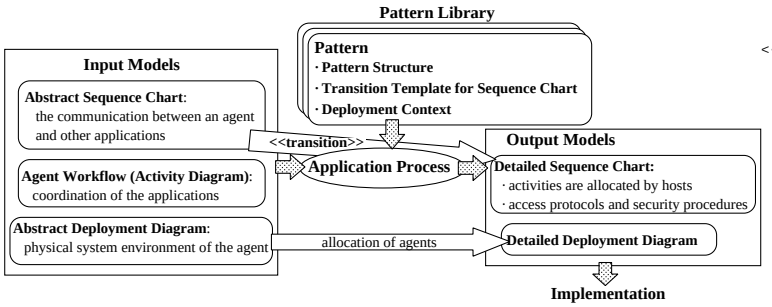


Figure 1. Overview of Our Method

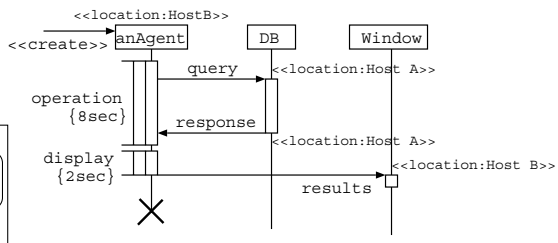


Figure 2. An Example of Abstract Sequence Chart

an agent and other applications. The **Agent Workflow** describes the coordination of the applications using activity diagrams. The **Abstract Deployment Diagram** defines the physical system environment of the agent.

2. Application of patterns.

   We build the detailed system behavior specifications that meet the models by instantiating the patterns into system behavior fragments. The pattern library consists of several patterns. Each pattern consists of a **Pattern Structure**, a **Transition Template** for the Sequence Chart that defines the applicable part of patterns and a **Deployment Context** that constrains the pattern application.

   A **Transition Template** consists of an applicable part and the detailed chart and illustrates how to rewrite the **Abstract Sequence Chart** to make it detailed. The rewrite of the abstract one is defined using a transition of the applicable part into the detailed one. The output model rewritten by **Transition Templates** is called the **Detailed Sequence Chart**, which includes access protocols and security procedures. In the **Detailed Sequence Chart**, all activities are allocated by hosts. Agents introduced in the access protocols are allocated to the **Abstract Deployment Diagram** at the application process. The deployment diagram in which all agents are allocated is called **Detailed Deployment Diagram**, which is another output model.

3. Selection of the specifications from the candidates with respect to the total performance data.

   Only one final specification can be selected from among the candidates. We calculate the total time including the network communication time and the time for the security measures of the candidates and select the one with the least total time. By this means, the developer can choose the most appropriate implementation. Once this has been done, designers can implement the system on the basis of the output models.

3.2 Input Models

In our approach, we use the following three kinds of input models: Abstract Sequence Charts, Abstract Deployment Diagrams and Agent Workflows. These models express an agent specification using established UML notation.

An **Abstract Sequence Chart** is a model expressing message interchange of applications through an agent. The model is annotated by computation time for agent actions and where the computation takes place. This time is used to calculate total computation time for the agent. Figure 2 shows an example of an Abstract Sequence Chart. In this chart, a box indicates a system component and an arrow to a box with ≪`create`≫ means that the component is created by the system at the beginning. In this example, `anAgent` is created by a system and queries the `DB`, then does some computation and sends the results to `Window` as a bitmap image.

The computation time of actions is expressed as timing constraints of UML. For `operation` and `display` in our example, these are 8 sec. and 2 sec., respectively. The stereotype ≪`location`≫ indicates the location of the message sender or receipt host. In our example, `anAgent` is created at `host B`. In addition, `DB` is located at `host A` and `Window` application is at `host B`, because the destination hosts of `query` message and `results` message are `host A` and `host B`, respectively.

In this model, each computation host need not be fixed. The undefined part should be fixed at the **Application Process**. In our example, the `operation` and `display` activities are not fixed to any hosts. The activities are assigned to hosts, `Host A` or `Host B`, at the **Application Process**. The host information is used by Transition Templates to allocate the agent actions to appropriate hosts.

An **Abstract Deployment Diagram** expresses the physical location of applications and an agent, and network and host's information. It includes computation nodes, network bandwidth, and access control information for the hosts and the network. Access control information and network speed

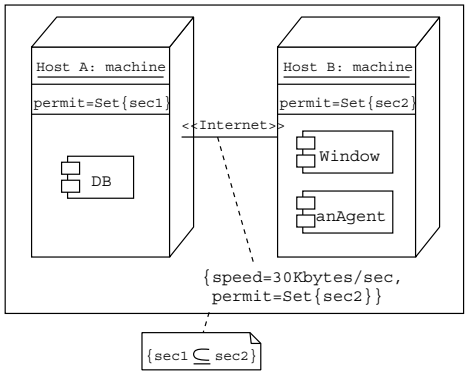


Figure 3. An Example of Abstract Deployment Diagram

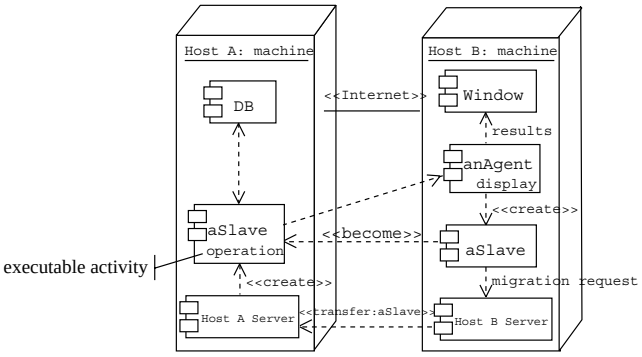


Figure 5. An Example of Detailed Deployment Diagram

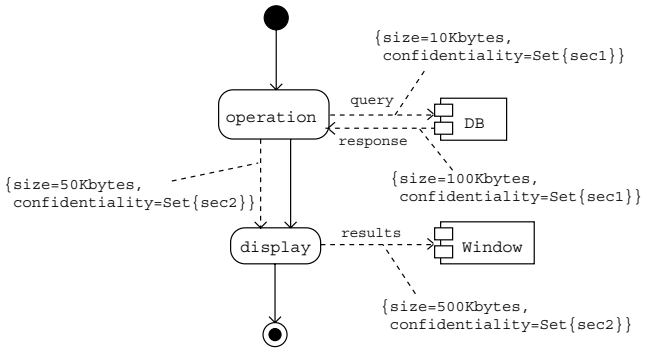


Figure 4. An Example of Agent Workflow

are expressed by properties named "permit" and "speed", respectively. Figure 3 shows an example of an Abstract Deployment Diagram. In our example, there are two applications named DB and Window, and one agent named anAgent. The DB is on Host A. The Window and anAgent are on Host B initially. A permit property of a host or a network indicates who can access the host or the network to use it. For example, permit=Set{sec1}, a property of Host A, indicates that only the members of sec1 can access Host A. In addition, the members of sec2 can access Host B and the network between Host A and Host B. sec1 and sec2 are control domain names. According to restriction sec1 $\subseteq$ sec2 in the figure, the members of sec1 can also access Host B and the network.

speed property of a network indicates the available bandwidth of the network for communications of an agent. For example, the figure 3 shows available bandwidth of the network between Host A and Host B is 30 Kbytes/sec. The access control information is used when the pattern condition is checked. The network bandwidth is used for calculation of the messaging time.

An **Agent Workflow** is a model expressing coordination of the agent and other applications. This model is represented as an Activity Diagram. It includes message flows between the agent and applications and intermediate data flows of the agent, the data size, and confidentiality information. The data size and confidentiality information are expressed by properties named "size" and "confidentiality", respectively. The confidentiality property attached to a data flow indicates who can be permitted to read it. Figure 4 shows an example of the Agent Workflow. A dotted line indicates message flow or intermediate data flow. Data flows between an agent and application are message flow, and data flows between agent activities are intermediate data of the agent. In this case, the agent proceeds to operation and display activities. The data flow from operation to display is an intermediate data flow of the agent. The size of an input data of display activity is 50 Kbytes and only members of sec2 are permitted to read it. Similarly, the members of sec1 are permitted to read the query and response messages to DB in addition to the input data of display.

Confidentiality properties of this model are used to check a pattern condition with permit properties of an Abstract Deployment Diagram. In addition, size properties are used to calculate messaging time using bandwidth information in the diagram.

3.3 Output Models

Output models consist of a Detailed Sequence Chart and a Detailed Deployment Diagram. In a **Detailed Sequence Chart**, every activity is allocated to particular host and access protocols of applications and security implementation are fixed. The access protocols may include remote messaging, cloning and migration of an agent. Security implementation may include authentication, encryption/decryption, signature/verification of messages and key exchange protocols. The right side of Figure 11 is an example of a Detailed Sequence Chart. In this case, anAgent and DB exchange

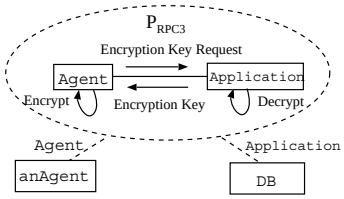
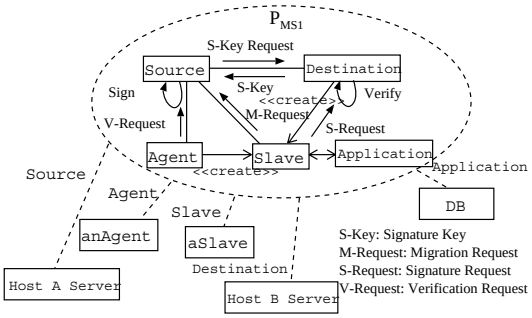


Figure 6. Pattern Structure of P_{RPC3}



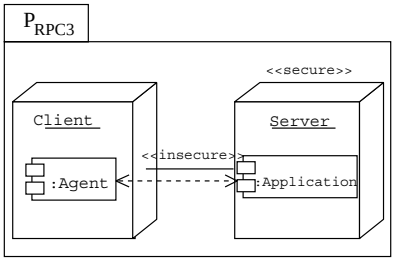


Figure 8. Deployment Context of P_{RPC3}

Figure 7. Pattern Structure of P_{MS1}

an encryption key at beginning to encrypt messages between them. In other words, `query` and `response` message is encrypted by `anAgent` or `DB`. The encryption and decryption activities are indicated by "Encrypt" and "Decrypt" in a sequence.

The other output model is a **Detailed Deployment Diagram**. This model comprises all agents, including cloned ones. All agents and activities are allocated to particular hosts. Figure 5 is an example of a Detailed Deployment Diagram. In this case, `DB` is accessed by `aSlave` agent created by `anAgent`. Allocated activities are expressed by executable activities of an agent, which is listed in an agent component box in the diagram. In the case of Figure 5, `operation` activity is executed by `aSlave` agent on `Host A` and `display` is executed by `anAgent` on `Host B`.

In addition, migration of an agent is expressed using stereotype ≪`transfer`≫, ≪`create`≫ and ≪`become`≫. When an agent tries to migrate to another host, the agent, first, requests the agent platform to migrate. The platform, then serializes the agent to send it to a platform of another host, which is expressed using ≪`transfer`≫ of the agent. Finally, the platform receiving the agent deserializes and invokes it, which is expressed using ≪`create`≫ of the agent. In Figure 5, `aSlave` agent migrates from `Host B` to `Host A` through `Host B`'s platform named `Host B Server` and `Host A`'s platform named `Host A Server`. Furthermore, cloning of an agent is expressed using ≪`create`≫. To put it precisely, cloning agent (master agent) copies its state and a part of its intermediate data and code, and then invokes the copied agent (slave agent), which is expressed using an arrow from a master agent to a slave agent with ≪`create`≫. In the case of figure 5, `anAgent` creates a clone named `aSlave` on `Host B`.

3.4 Pattern Library

A pattern library consists of three definitions: Pattern Structures, Deployment Contexts and Transition Templates for Sequence Chart. The Pattern Structures describe the parameters of the pattern. The Deployment Contexts describe the physical condition of the pattern. Transition Templates illustrate how to get a Detailed Sequence Chart from an abstract one. All patterns are expressed using a common template appearing in *desgin pattern*[1]. However, we illustrate only concepts and provide a brief description of the patterns in this section.

A **Pattern Structure** expresses the collaboration of agents and applications. This structure includes components which indicate agents and applications, relations and messages between the components. The components are the parameters of this pattern. The parameters are replaced by the actual components of a system if the patterns are applied. Figures 6 and 7 show some of the Pattern Structures. In the case of Figure 6, `Agent` and `Application` are parameters, so that the parameters are replaced by the actual components named `anAgent` and `DB`, respectively.

A **Deployment Context** in the pattern library constrains pattern application. This model is a diagram which expresses a context of components appearing in Pattern Structure. All components in Pattern Structure are placed on hosts in the diagram. In addition to it, this diagram includes some restriction indicated by stereotype: ≪`secure`≫ and ≪`insecure`≫. The restriction is attached to a network or a host. The network is between agents or between the agent and an application. The host is also the destination for the migrating agent. A network line with ≪`secure`≫ indicates that the network is secure. The stereotype depends on the access control of the network and confidentiality of the data through the network.

Figures 8 and 9 are examples of Deployment Contexts of patterns. Figure 8 shows that the network between the `Client` and the `Server` is insecure. It implies that the message between the `Agent` and the `Application` may be leaked to inappropriate persons who are not permitted to read the message. In this case, the `Server` is secure because the message is not leaked to inappropriate persons on the `Server`. How we establish this is illustrated in Sec-

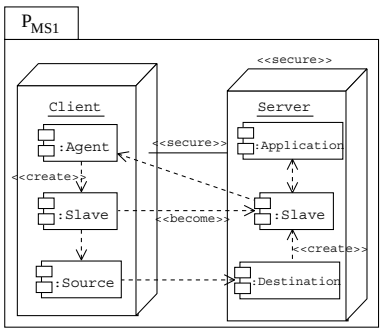


Figure 9. Deployment Context of P_{MS1}

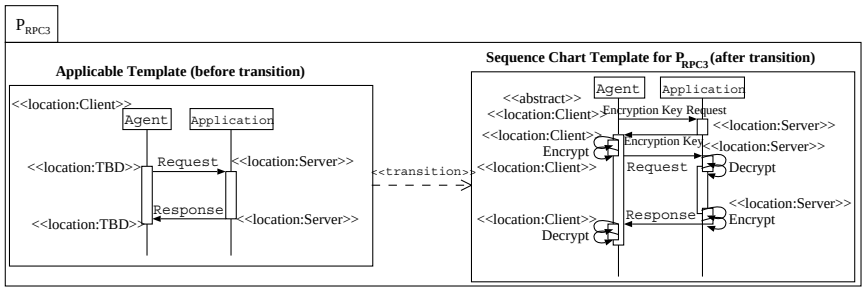


Figure 10. Transition Template for Sequence Chart

tion 3.5. In the case of Figure 9, the network between the `Client` and the `Server` is regarded as secure, because both the data which migrating `Slave` agent holds and the message between the `Slave` and the `Agent` are not leaked.

We currently have twelve patterns. The patterns from P_{RPC1} to P_{RPC4} are for the remote messaging protocol. These patterns denote that the host performing the computation interacts by exchanging messages between the host that has the applications and the data related to the computation. The patterns from P_{MS1} to P_{MS4} are for master/slave protocol. These patterns denote that an agent performs the computation in question and the next computation in different hosts. In this pattern, the mobile agent migrates partially or as a whole to the host for the next computation. The original mobile agent is blocked until the remote computation is completed. The patterns from P_{MO1} to P_{MO4} are for mobile agent protocol. These patterns denote that an agent performs the computation in question and the next computation in different hosts, and the agent migrates as a whole.

The patterns take into account encryption and decryption for preventing data leakage, signature generation and verification for preventing tampering, and authentication for preventing masquerading. From this point, we call our patterns **security patterns**.

The time for encryption, signature, and authentication are included in the patterns. Thus, the time for encryption is included in P_{RPC3} and P_{RPC4} in which the keys are exchanged via an insecure network. The time for authentication is included in P_{RPC2} and P_{RPC4} in which one of the hosts is insecure. In this paper, all the data flowing in the network are given signatures even in P_{RPC1} and P_{RPC2} in which the network is secure, because we assume that there is a danger of tampering even if the network is secure. The cases of P_{MSn} and P_{MOn} include the time of those security measures in the same way. The following table relates the application conditions, patterns, and the security measures.

host	network	applicable pattern	security
secure	secure	P_{RPC1}, P_{MS1}, P_{MO1}	signature
insecure	secure	P_{RPC2}, P_{MS2}, P_{MO2}	auth.&signature
secure	insecure	P_{RPC3}, P_{MS3}, P_{MO3}	enc.
insecure	insecure	P_{RPC4}, P_{MS4}, P_{MO4}	auth.&enc.

Where "auth." and "enc." indicate authentication and encryption measure, respectively.

This table shows P_{RPC1}, P_{MS1} and P_{MO1} as the patterns to be applied in the case where network is secure and the host with application is secure. The definition of secure or insecure is shown in Section 3.5.

A **Transition Template** consists of an applicable template (before transition) and a sequence chart template (after transition). An applicable template denotes to which part the pattern is applied in an Abstract Sequence Chart. An applicable template is connected with the sequence chart template which illustrates the detailed sequence after application of the pattern. The pattern application of a sequence chart is expressed by the transition indicated by an arrow labeled stereotype $\ll$`transition`$\gg$. The translated templates include an access protocol and security procedures.

We should make the Detailed Sequence Chart to implement a target system. When we apply the Transition Templates to an Abstract Sequence Chart, we can get the rough copy of detailed one of the target system. The Transition Templates include some parameters. To make the Detailed Sequence Chart complete, we should use the Pattern Structure and the Detailed Deployment Diagram. First, we relate the components in the Pattern Structure with the instances of the target system. Component names in the Transition Templates correspond to the instances. Secondly, we allocate the instances to the hosts in the Abstract Deployment Diagram of target system using a Deployment Context of the pattern. Message names and host names in the Transition Templates should be rewritten by instance name in the Abstract Sequence Chart or the Abstract Deployment Diagram of the system.

Figure 10 illustrates Transition Templates of P_{RPC3} and P_{MS1}. In this figure, the label named "TBD" indicates that

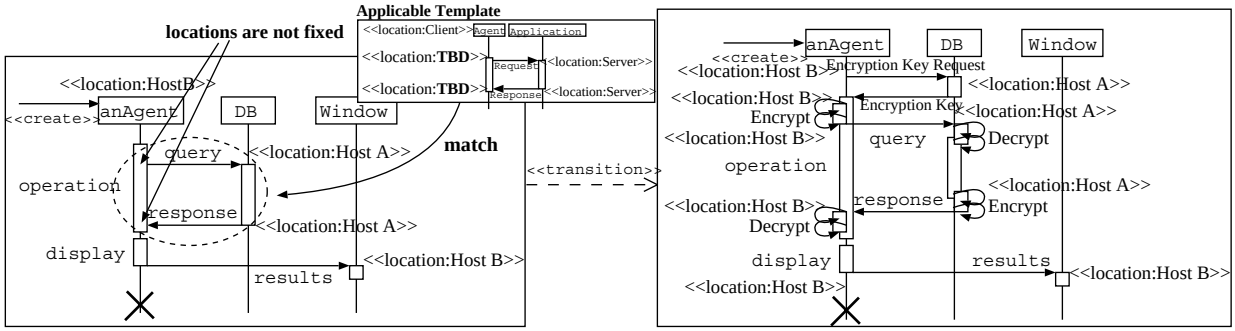


Figure 11. An Example of Sequence Chart Transition

the location is not fixed, so that the Transition Templates allocate the locations to the labels. The case of P_{RPC3} includes encryption key[1] exchange protocol for encryption of the `Request` and `Response` messages. In addition, it allocates the source host of the `Request` message and the destination host of the `Response` message to `Client`. The key exchange protocol is abstract, so that the protocol is instantiated by security implementation of a target system. The case of P_{MS1} includes master/slave protocol and verification of the messages. In the case, `Agent` creates a `Slave` agent, then `Slave` migrates to `Server` to use `Application`. The `Response` message received by `Slave` agent is forwarded to the original mobile agent, `Agent`, which is blocked until the remote computation is completed. In addition to it, this case includes signature key exchange and signing of messages through the network between `Client` and `Server` to verify them. The messages include intermediate data of the `Slave` agent.

3.5 Pattern Application

In this section, we explain the application process by using input models shown in Figures 2, 3 and 4. In the process, we use a **Pattern Structure**, a **Transition Template**, and a **Deployment Context** of the pattern to apply the pattern in the pattern library to these input models. By the process, we can make a detailed specification from the abstract one (Figure 11).

At the first step of application, we look for applicable part in **an Abstract Sequence Chart** to enumerate available patterns. To put it more concretely, we search parts at which sender or receiver locations are not defined in **an Abstract Sequence Chart**. Then, we check that each part matches the **Applicable Template**(the left side of **Transition Template**), in which undefined locations correspond with ≪`location:TBD`≫.

For each available pattern, we look for correspondences of the parameters in the **Applicable Template** and the instances in **an Abstract Sequence Chart**. In the **Applicable Template**, locations, message names, and component names are parameters. In our example, the `operation` activity of Figure 2 matches the **Applicable Templates** in Figure 10, because the source host of `query` and the destination host of `response` message are not fixed. In addition, the parameters named `Client` and `Server` in the **Applicable Templates** correspond to `Host B` and `Host A` in Figure 2, respectively (the left side of Figure 11). In this case, `anAgent` and `DB` components correspond to `Agent` and `Application` parameters in the pattern library.

At this point, more than one pattern can be available for application. For example, both P_{RPC3} and the P_{MS1} are available. We illustrate some examples of the application process using these patterns.

Next, we select applicable patterns from the enumerated patterns at the above step. First, we check the application condition of a **Deployment Context** in the pattern library using an **Agent Workflow** and an **Abstract Deployment Diagram**. Figure 12 illustrates the details. This figure summarizes the case of P_{RPCn}. Similarly, we can check the application condition of P_{MOn} or P_{MSn}. This step uses the confidentiality information from an **Agent Workflow** and the access control information from an **Abstract Deployment Diagram**. The application condition of the **Deployment Context** of the pattern is a security issue denoted by a stereotype ≪`secure`≫ or ≪`insecure`≫ ((c) in Figure 12).

The following is how we decide if the system is secure or insecure. First, we temporarily instantiate the parameters of **Pattern Structures** ((d) in Figure 12). At this step, we only consider communications between agents and applications. The instances of agents and applications in **Pattern Structures** are the components in the **Abstract Sequence Chart** or new agents for the pattern. The correspondences of components in the **Abstract Sequence Chart** are considered. A new agent is introduced when an agent in the **Abstract Sequence Chart** creates a clone agent in the pattern. In the case of Figure 7, the instances of `Agent` and `Application` are `anAgent` and `DB`, respectively. In ad-

[1]Encryption keys are RSA public keys or shared keys encrypted by RSA keys.

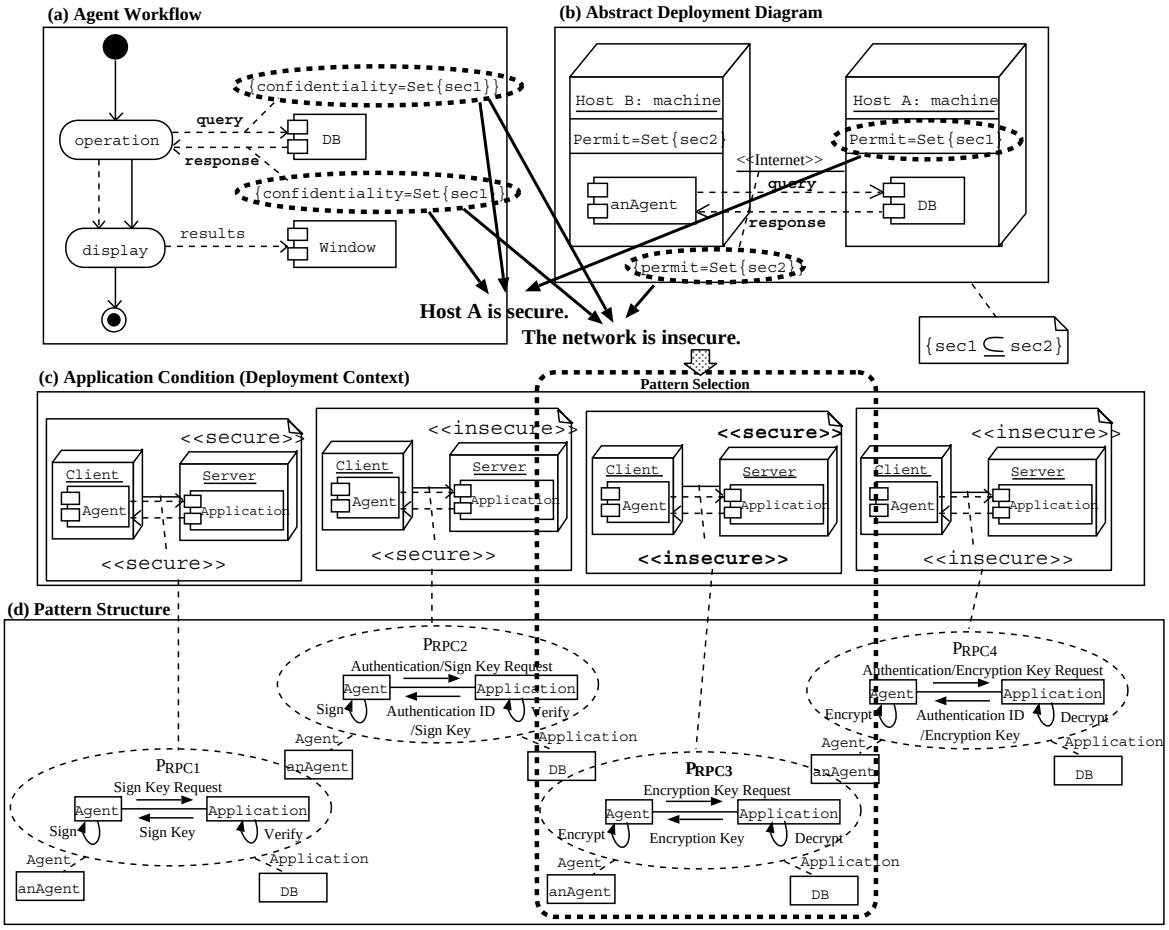


Figure 12. An Examination of Application Conditions and A Pattern Selection

dition, a new agent named `aSlave` is introduced, which is created by `anAgent`.

Secondly, we allocate the agents in the **Pattern Structure** to the **Abstract Deployment Diagram**. In the case of Figure 12, `anAgent`, which is the instance of Pattern Structures in (d), is allocated to the Abstract Deployment Diagram labeled by (b) in the figure. Then, we check the message flows passing through the network. In the case of (b) in the figure, `query` and `response` messages are passing through the network. Finally, we find the confidentiality properties of the message flows in the **Agent Workflow** and the permission properties of the `Server` host and the network in the **Abstract Deployment Diagram**((a) and (b) in Figure 12). In the example illustrated in Figure 12, both `confidentiality` properties of `query` and `response` messages are a set of `sec1`. The `Permit` property of the network is `sec2`. Furthermore, the `Server` parameter in the **Deployment Context** of P_{RPCn} corresponds to `Host A`, and the host's `Permit` property is `sec1`.

We can define the secure of a network and a host using the properties as follows.

Definition 1: A network is secure if $a \supseteq b$, where a is a joined set of confidentiality of the flows, and b is a set of access control domains of the network, which is indicated by `Permit` property of the network. Otherwise, the network is insecure.

Definition 2: A host is secure if $a \supseteq c$, where c is a set of access control domains of the host with the application, which is indicated by `Permit` property of the host. Otherwise, the host is insecure.

In the case of our example, host A is secure, because the joined set of confidentiality of `query` and `response` is `sec1`, corresponding to a in the **Definition 2**, and an access control domain of the host is `sec1`, corresponding to c in the definition. Similarly, the network between `Host A` and `Host B` is insecure, because an access control domain the network is `sec2` corresponding to b in the **Definition 1**. Hence, we can apply the P_{RPC3} to our example.

At the next step, we apply the pattern selected at the previous step to the input models. First, we instantiate the parameters of the applicable **Pattern Structure** into the cor-

responding components of the target system. Figures 6 and 7 are instances of P_{RPC3} and P_{MS1}, respectively. All instances of agents and applications become clear at the previous steps. In this step, we must make the correspondence of agent platforms in a system. For example, `Source` and `Destination` in Figure 7 correspond to `Host A Server` and `Host B Server`,respectively. These components are agent platforms of `Host A` and `Host B`.

We use these patterns, P_{RPC3} and P_{MS1}, to describe the following steps. Although we illustrate only two applications of Pattern Structure in this paper, we can instantiate other Pattern Structures.

Next, we allocate the components which are parameters in the **Pattern Structure** to particular hosts in the **Abstract Deployment Diagram**. The location of the hosts is based on the **Deployment Context** of the pattern. When we apply the P_{MS1} to our example, we can allocate `aSlave` agent in Figure 7 to `host A` and `host B` in Figure 3, because the instances of `Client` and `Server` in Figure 9 are `host B` and `host A`, respectively, and `aSlave` is on these hosts. Figure 5 is an example of allocation of the components appeared in Figure 9.

At the final step, we rewrite the **Abstract Sequence Chart** to get a detailed one. First, we replace all the parameters of a **Transition Template** by the corresponding instances. Almost all parameters are fixed at previous steps. At this point, we decide a security implementation. For example, authentication protocols and encryption / decryption procedures are added. Finally, we make a **Sequence Chart Transition** diagram by applying the instance of the **Transition Template** to the applicable sequence. Figure 11 is an example of a **Sequence Chart Transition** for the application of P_{RPC3} in Figure 10 to our example (Figure 2). In Figure 11, the `operation` part is rewritten by a Sequence Chart Template of P_{RPC3} to make it detailed. In other words, the execution host of `operation` is fixed to `Host B` and the access protocol to `DB` is fixed. In addition, encryption and decryption of messages and encryption key exchange protocol are added.

In the same way, we can consider the application of the other patterns P_{MSn} and P_{MOn}.

For selection of the output models, we calculate the performance of the models, which is described in Section 3.6.

3.6 Detailed Specification Selection

Once the process described in Section 3.5 has been completed, there are usually several candidates of detailed specifications of the system behaviors. We have opportunities to create more than one candidate, because more than one pattern condition are satisfied. In order to choose the final specification from the candidates, we calculate the total time consisting the computation time (CT), the message communication time (MT) and the time for security measures (SM) of the candidates by the following formula. MT includes the migration time of agents.

$$\texttt{Totaltime} = \Sigma\texttt{CT} + \Sigma\texttt{MT} + \Sigma\texttt{SM} \quad (1)$$

Using this formula, we select the one with the least time.

Table 1. The Time for Security Measure of Each Pattern

Pattern	Time for Security Measure
P_{RPC3}	K-ex $+ n \times$ (encryption $+$ decryption)
P_{MS1}	K-ex $+ 2 \times$ (signature $+$ verification)
P_{MO4}	authentication+K-ex+encryption+decryption

Here, the time for security measure is the summation of the following: the time for authenticating the communicating hosts, the time for obtaining the encryption/signature keys, the encryption and decryption time, and the signing and verification time.

In actual total time calculation, since the detailed specifications are composed of pattern instances, we first calculate the time of each instance. Table 1 shows the formulae for the time calculation for security measure of the patterns P_{RPC3}, P_{MS1} and P_{MO4}. In this table, "K-ex" and "n" indicate the key exchange time for encryption or signature of messages, and the frequency of messages between an agent and an application related to each pattern, respectively. Pattern P_{RPC3} includes key exchange time for encryption, message encryption time and message decryption time by the agent and the application. Similarly, P_{MS1} includes the signing and verification time for security. In addition to P_{RPC3}, P_{MO4} includes the time for authentication protocol. The time is indicated by "authentication" in the table. In the case of P_{MO4}, intermediate data of the agent should be encrypted.

4 Case Study and Evaluation of our approach

4.1 Case Study of the ECP Design Support System

The ECP design support system helps designers of products such as electric household appliances and office machines to take into consideration environmental impacts of the products throughout their lifecycles. From the product designers' viewpoint, the actual implemented system is realized as a single add-on function for CAD. During the product design phase, the designers can use the ECP design support system to evaluate the environmental impacts a contemplated product would have. The ECP system includes

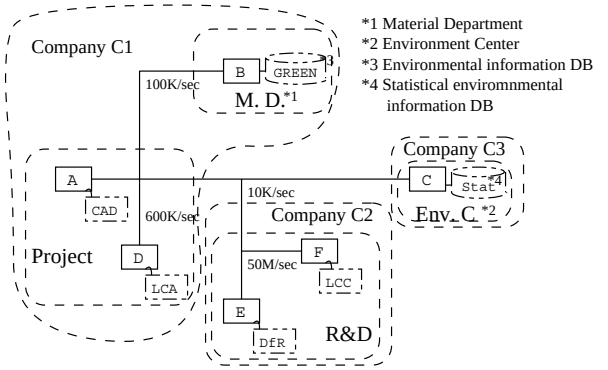


Figure 13. Environmentally Conscious Product (ECP) Design Support System

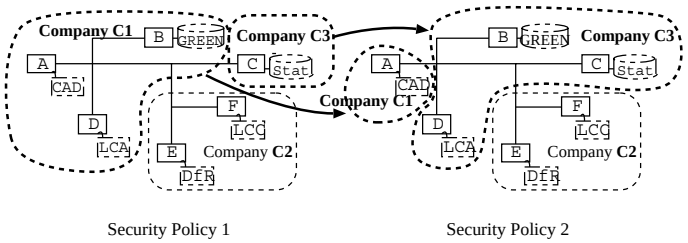


Figure 15. Security Policies for the ECP Design Support System

some kinds of environmental information and tools belonging to some companies. In detail, as shown in Figure 13 it is necessary to coordinate CAD, the simulator (LCA: LifeCycle Assessment tool) and the GREEN DB for the product parts in Company C1, the two simulators (DfR: Design for Recycle tool, and LCC: LifeCycle Cost calculation tool) in Company C2, and the Stat DB in Company C3. The environmental information on a product part includes such data as how much harmful material escapes to the environment from the production phase to final disposal. The companies involved are connected via a network whose bandwidth is narrower than that of a network inside a company, and is more susceptible to unauthorized access.

We implemented the flow of the target service in the following way. First, the system retrieves the environmental information and the statistical information on the parts of the contemplated product. Then, the system calculates the environmental impacts of the product by using the simulators of Company C1 and Company C2 and finally displays the results to the designer.

We already have the services and the components needed for the system. The items that need to be newly developed for the system include security facilities and coordination of services and components.

4.2 Evaluation of Our Example

In this section, we evaluate our method by designing the system described in Section 4.1. Table 2 shows the computation time including access protocols and security procedures. For our evaluation, we define two types of security policy (Figure 15). Security Policy 1 is expressed by the configuration described in Section 4.1. The case of Security Policy 2 is that the GREEN DB and LCA, which are under the control of Company C1, are changed to control of Company C3.

For our system, the security procedure implementation is as follows.

- A common certification authority is prepared for the server authentication between two different companies.
- An encryption key is generated for every lifecycle of the agent.
- A single signature key is used during the entire period that the system is operating.

By measuring the actual time for each security procedure, we obtained the following figures as the average time.

- The verification key exchange time: 0.0 sec.
- The authentication time (AT) inside a company (ATi): 1.6 sec.
- AT between companies (ATo): 5.0 sec.[2]
- The decryption key exchange time (DKE) inside a company (DKEi): 1.0 sec.
- DKE between companies (DKEo): 1.5 sec.
- The time for encryption and decryption (ET): 1.0 sec. per message
- The signature and verification time (VT): 0.7 sec. per message
- The migration time[3] for an agent : 0.1 sec.

For Security Policy 1, we compare the two models: Model A and Model B, which implement different behavior. These simplified behaviors of a part of our system are illustrated in Figure 14. This figure illustrates that a slave agent in Model A created by an agent migrates to Host C to use Stat database locally. In the case of Model B, an agent makes the round of hosts, Host B, Host C, and Host E, to access the resources locally.

Model A is more efficient than Model B, because the agent in Model B includes unnecessary migrations with important data. In addition, the behavior requires more security considerations than the case of Model A. Model A includes three remote messagings for two P_{RPC1} and P_{RPC4},

[2] The time for certification authorities is included.

[3] The time includes serializing and deserializing time for the migration.

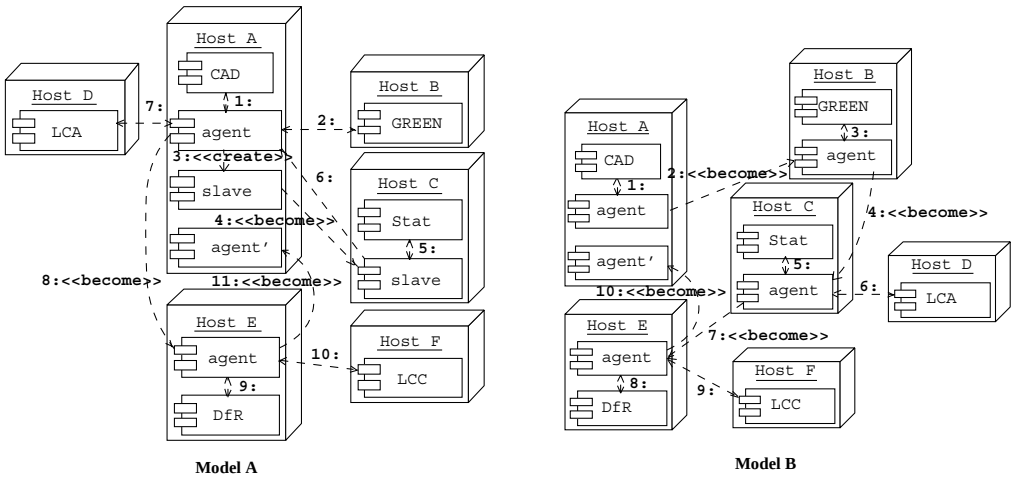


Figure 14. Simplified Deployment Diagram of Model A and B

Table 2. Comparison of Models

Time(sec)	Model A	Model B	Model A'	Model B'
Message/Migration	6.22 (w. 3 migrations)	8.2 (w. 4 migrations)	6.22 (w. 3 migrations)	8.2 (w. 4 migrations)
Security	26.4	31.7	40.6	27.8
Total Time	109.8	117.1	124.0	113.2
Applied Patterns	$P_{RPC1}\times2$,P_{RPC4}, P_{MS4}, P_{MO3}, P_{MO4}	P_{RPC3},P_{RPC4}, P_{MO3}, $P_{MO4}\times3$	$P_{RPC4}\times3$, P_{MS4}, P_{MO3}, P_{MO4}	$P_{RPC4}\times2$, P_{MO3}, $P_{MO4}\times3$
Security Measure	ATi, ATo×2, DKEi, DKEo×2, ET×8, VT×2	ATi×3, ATo, DKEi×3, DKEo×2, ET×12	ATi, ATo×4, DKEi, DKEo×4, ET×12	ATi×4, ATo, DKEi×4, DKEo, ET×12
Security Policy	Policy 1		Policy 2	

and three migrations of agents for P_{MS4}, P_{MO3} and P_{MO4}. Table 2 indicates that model A includes 6.22 seconds for the time of the messaging and the migrations. In addition, model A includes three authentications for P_{RPC4}, P_{MS4} and P_{MO4}, verification time for P_{RPC1}, and encryption time for P_{RPC4}, P_{MS4}, P_{MO3} and P_{MO4}. The time for security measures is 26.4 seconds. The total time of model A is 109.8 seconds, including computation time of other activities (77.2 sec.). In the case of Model B, it includes four migration time. In the case of Policy 1, Model A is more efficient than Model B (A:$109.8 <$ B:117.1).

Model A' and Model B' are in the case of Security Policy 2, in which a security policy is different from that of Model A and Model B. Other definitions in the system input models are the same as Model A or Model B. The behavior of these models is the same as Model A or Model B except for security implementation. For changed security policy, the patterns applied are changed. As a result, the efficiency of models is changed.

In this case, if we consider only the messaging and migration time, Model A' is more efficient than Model B', because the messaging/migration time of Model A' is smaller than the case of Model B' (A':$6.22 <$ B':8.2). However, Model B' is more efficient than the other, if we consider the security. In other words, Model B' includes less time for security measure than the other (A':40.6>B':27.8). Consequently, Model B' is more efficient than Model A'.

The security measure is determined by security policy. It implies that the efficient models depend on the security policy and its implementation.

4.3 RPC v.s. Mobile Agents

In this section, first, we analyze our models described in Section 4.2 in order to discuss the characteristics of mobile agents. Moreover, we take up the security concerns of mobile agent systems.

As described in section 4.2, the message/migration time of Model A/A' is shorter than the time of Model B/B'. In Model B or B', the agent communicates with an application through a narrow network using big data. In detail, in Model A, the agent accesses the LCA simulator through the intranet of Company C1 between hosts A and D, the bandwidth of which is 600 Kbytes/sec. In this case, the communication takes 0.03 sec. because the message size is 18 Kbytes. In contrast, the agent in Model B accesses the simulator through the extra-net between Host C in Company C3 and Host D in Company C1, the bandwidth of which

is 10 Kbytes/sec. Thus, the communication takes 1.8 sec. That is one of the reasons why Model B/B' includes longer time than the other. However, the difference is small in our example, and so we can disregard the difference.

In general, an inter-company coordination system includes narrow-band networks between companies and high-speed networks within a company. Therefore, we cannot always neglect the difference between the messaging through the narrow-band network and the case of high-speed one. Besides the messaging, we should consider the case of an agent migration with big intermediate data through a narrow-band network.

From the viewpoint of security, Model B includes more security than Model A in the case of Policy 1. This is because the agent in Model B moves around the companies with product design information, which is important and can be opened to only members of the project. This leads to additional security measures, namely, authentication and encryption between companies. The security measures across companies take a longer time than the case of intranets. Therefore, we should reduce the security measures across companies. By migration of a mobile agent, we can reduce the extra security. For our example of Policy 2, an agent in Company C1 should access some DBs, `GREEN` and `Stat` in Company C3 through the extra-net and the communication is private for Company C1. In this situation, the migration of the agent to Company C3 requires only one authentication between companies, although the remote access requires two authentications between companies. In our example, the difference of security time is 5.3 sec. in the case of Policy 1 and 7.8 sec. in the case of Policy 2. We cannot ignore the differences, because it takes 6.9% or 10.1% of total activity.

Using our method, we only consider guarding an agent itself against the other software. Therefore, in addition to our method, we must consider protection of hosts in a system in order to develop the system. Although mobility of an agent may reduce the number of security measures, the technology includes new types of security risks. For example, acceptance of a malicious agent can give rise to propagation of system violations in the company. However, our method can be applied to an inter-company distributed system, in which it is easier to confirm an agent code than in the case of an open distributed system.

In addition to the efficiency reason, code mobility makes it easier to maintain the software components across companies than distributed objects statically. In the case of our example, by using the mobile agent technique, we can separate the maintenance of an agent calculating environmental impacts and the maintenance of the simulators in another company. However, the migration of an agent is not always more efficient than remote messaging, and so we must choose the appropriate techniques.

4.4 Evaluation of Our Approach

We examined our approach as follows by developing the ECP design support system and adjusting the system according to the changes that may appear in the system.

We limited the implementation techniques to the two, that is, remote messaging and mobile agents, and prepared the patterns. However, the number of choices is very large even with only these two techniques and it is very difficult to select appropriate implementations since the mobile agents are not always better than remote messaging. As for our ECP design support system, multiplying the number of choices for each part of the dataflow reveals that there are more than one thousand possible choices.

On the other hand, we defined the performance and the security time of the system and treated them as the criteria of pattern selection in our approach. As a result, even inexperienced developers can dispose of the inappropriate candidates and select the appropriate ones. As for the example in Section 4.1, experience is not required in order to select the candidate Model A or Model B' by calculating the security time that are objective values.

The systems should be kept free of security holes. However, it is difficult to check the existence of security holes completely. For example, if we try to realize security in the system behaviors of Figure 14, it is difficult to decide which parts need authentication or encryption. Our approach makes it possible for developers to select security patterns that have appropriate guidelines about authentication and encryption. The guidelines of selection were explained in Section 3.5, where the `confidentiality` property in **Agent Workflow** and the `permit` property in **Abstract Deployment Diagram** are used.

It is difficult to realize security efficiently. As for the example in Section 4.1, Model A' is more efficient without security than Model B', whereas the Model B' is more efficient if security is considered. According to Table 2, Model A is more efficient totally than Model B in the case of Policy 1. However, in the different case, Policy 2, the latter behavior named Model B' is efficient than the former one, Model A'. The security overhead of our system is about 10 percents of the total turnaround time of the service. Therefore, it cannot be neglected in the effort to achieve an efficient system design. Since it is difficult even for experienced developers to design appropriate specifications that take the security overhead into consideration, our approach comes into its own in such situations.

4.5 Assumptions and Limitations of our Approach

The following conditions are prerequisites of our approach.

- The resources including application software, databases, and hardware configurations are explicitly known to the developers. In addition, the constraints of the resources such as the network performance and the computational efficiency of the hosts are known in advance.
- APIs to existing applications and databases are available locally on the hosts or remotely via the network. Therefore, the developers are ready to compose such software parts in order to build the target system.

  These two assumptions have become practical recently due to legacy software reuse techniques, which allows us to analyze the constraints of legacy resources, and component-based development techniques.

Our approach is subject to the following limitations.

- *It is essentially difficult to find the most appropriate implementation.* This is true especially for large systems. The reason is that the selection of implementations leads to combinatorial explosions. However, it is possible in practice to select relatively efficient implementations by, for example, decomposing the system or imposing some limitations on the system. In addition, we can make CASE tools including a pattern library for pattern application. Moreover, for large systems separation of concerns is key issue, and our models accomplish the separation.
- *The models are simple.* In some cases, it may not be easy to apply our approach since our system modeling is simplified. However, our main purpose in this paper is to consider general methodologies in which patterns are applied according to the system requirements and appropriate pattern combinations are selected by comparing the performance. Therefore, other cases could be dealt with by enriching the parameters of the models and preparing new patterns. Patterns should not break application logic, so that if new patterns are complex, we must verify the correctness. We can define appropriate equivalence between the applicable part and the rewritten part in each pattern up to some transitivity rules for the verification.
- *It is usually difficult to find accurate models.* System modeling is usually difficult even with the simple modeling of our approach. Such difficulties include the definitions of the amounts of data and the execution speeds. However, it is not a problem if these values are inaccurate because our aim is to examine which implementations are efficient in practice, not using absolute values but by means of relative comparisons of the values. We can select reasonable implementations by considering only predicted values, average values, or typical values.

5 Related Work

In this section, we compare our approach with some existing techniques in terms of the different viewpoints.

5.1 Patterns of Mobile Agent Applications

There are several other examples of research on agent design patterns such as [2]. However, most of this research does not consider security issues. Although [3] and [4] propose some security patterns, they are still at an early research stage and are not yet applied to any practical systems. On the other hand, this paper proposes a systematic and practical approach to distributed system development with mobile agents and patterns to realize security.

5.2 Selection of Implementation Techniques

Much work has been done on implementation techniques for distributed systems (messaging models). Messaging models can optimize the messaging time according to the application domains and the operation environments. For example, it is well known that messaging models are influenced by frequencies of message exchanges, message sizes, and network traffic. This makes it difficult to select message models in distributed system design.

Several results of research on messaging model selection have been reported [5, 6, 7, 8, 9]. In this research, the communication time of typical messaging models is compared by means of case studies and which model is efficient in which cases are discussed. [5] applies four models, namely, Client-Server, Remote Evaluation, Code on Demand, and Mobile Agent to a network management application and shows that the size of the codes transmitted through the network is the key parameter in selecting the models.

Our approach gives a systematic framework within which the viewpoints of this research can be applied by preparing patterns and their performance values explicitly. Our approach also adds the security viewpoints to the patterns. We expect to be able to apply these existing research results to refine our patterns, and thus realize more sophisticated guidelines for implementation selection.

5.3 Development Methods Considering Security

In general system development, nonfunctional aspects such as security are only considered after functional requirements are completely realized. However, it is difficult to realize secure systems with such design processes since the functional design developed in advance constrains the realization of security. Devanbu and Stubblebine [10] argue that it is necessary to deal with system models and security models at the same time for the optimal design of the whole

system with security and that this issue should be investigated in the future.

Our approach addresses the issue by dealing with the messaging models and the security models in the design phase. We consider security in calculating the communication time of the messaging models. As a result, we can give developers guidelines for model selection in which security is considered.

5.4 Development Methodology for Agent Systems

Much research has been done on development methodologies for agent systems [11, 12, 13]. The work propose methodologies to develop distributed systems by modeling the autonomous problem solving ability and the complex interactions of agents that cannot be handled by existing object-oriented methodologies.

Gaia [11] is a methodology that takes into consideration the organizational structures of agent systems and designs abstract models of agents in a top-down manner. This method can deal with simple access controls of agents and interaction patterns of agents. However, Gaia does not deal with security and policies. It lacks practicality in comparison with our work because, for example, it does not give any guidelines for selection of the interaction patterns.

MaSE [12] is an approach similar to Gaia. Although it has some limitations, for example, only one-to-one inter-agent interactions are allowed, it has an automatic code generation feature. AOR modeling [13] can specify inter-agent interactions formally by describing the interactions. However, these methodologies do not consider both of security policies and communication time.

6 Conclusions

Our aim is to propose a development method to realize secure and efficient distributed systems. In order to achieve this goal, some guidelines for selecting which implementation technique is more appropriate, remote messaging, cloning, or mobile agents, are required because it cannot be known in advance which technique is preferable according to the various conditions concerning security and performance. It is also difficult to realize security completely. Our approach solves these issues by using patterns that take security into account and providing explicit performance parameters for each pattern and selection procedures for the detailed specifications.

References

[1] E.Gamma, R.Helm, R.Johnson, and J.Vlissides. *Design Patterns*. Addison-Wesley, 1995.

[2] Y. Aridor and D. B. Lange. Agent design patterns: Elements of agent application design. In *Proc. of Agents'98*, 1998.

[3] Y. Tahara, A. Ohsuga, and S. Honiden. Agent system development method based on agent patterns. In *Proc. of ICSE'99*, pp. 356–367. IEEE, 1999.

[4] N. Yoshioka, Y. Tahara, A. Ohsuga, and S. Honiden. Security for mobile agents. *LNCS* 1957, Springer Verlag, pp. 223–234, 2001.

[5] A. Fuggetta, G. P. Picco, and G. Vigna. Understanding code mobility. *IEEE Trans. on Software Engineering*, 24(5),pp. 342–361, 1998.

[6] A. Carzaniga, G. P. Picco, and G. Vigna. Designing distributed applications with mobile code paradigms. In *Proc. of ICSE'97*, pp. 22–33. ACM Press, May 1997.

[7] M. Baldi and G. P. Picco. Evaluating the tradeoffs of mobile code design paradigms in network management applications. In *Proc. of ICSE'98*, pp. 146–155. IEEE, 1998.

[8] D. Deugo. Mobile agent messaging models. In *Proc. of ISADS 2001*, pp. 277–286, 2001.

[9] L. Ismail and D. Hagimont. A performance evaluation of the mobile agent paradigm. In *Proc. of OOPSLA'99*, pp. 306–313. ACM Press, 1999.

[10] P. T. Devanbu and S. Stubblebine. Software engineering for security: A roadmap. In A. Finkelstein, editor, *The future of Software Engineering*, pp. 225–240. ACM, 2000.

[11] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3),pp. 285–312, 2000.

[12] M. F. Wood and S.A. DeLoach. An overview of the multiagent systems engineering methodology. *LNCS* 1882, Springer Verlag, pp. 207–221, 2001.

[13] G. Wagner. Agent-object-relationship modeling. In *Proc. of 2nd Int. Symp. - from Agent Theory to Agent Implementation together with EMCRS*, 2000.