Computer Science Publications   Computer Science Department

2003

Policy Driven Licensing Model for Component Software

Zhao Qian
University of Western Ontario

Zhou Yu
University of Western Ontario

Mark Perry
University of Western Ontario, mperry@uwo.ca

Policy-Driven Licensing Model for Component Software

Qian Zhao, Yu Zhou, Mark Perry
Department of Computer Science
University of Western Ontario
London ON N6A 5B7 Canada
{ qianzhao | yuzhou | markp } @csd.uwo.ca

Abstract

Today, it is almost inevitable that software is licensed, rather than sold outright. As a part of the licensing policy, some protection mechanisms, whether hardware, legal or code-based, are invariably built into the license. The application of such mechanisms has primarily been in the realm of off-the-shelf, packaged, consumer software. However, as component-based software gradually becomes mainstream in software development, new component-oriented licensing systems are required. This paper proposes an enterprise component licensing model for the management of software component licenses. The model provides a comprehensive license management framework allowing for extensibility and flexibility. Furthermore, we identify differences between stand-alone software and component software, describe a high level model for policy-driven component licensing, and discuss both the benefits and drawbacks of the enterprise component licensing model for the management of software component licenses.

1. Introduction

The unauthorised copying, use and distribution of software, often labelled Software Piracy, is a continuing source of concern. Reports by the Software and Information Industry Association (SIIA) and Business Software Alliance (BSA) [1] indicate that annual losses for the business software application market due to software piracy were well over $12 billion for the year 2000, while SIIA calculated business software applications accounted for worldwide revenues of $21.6 billion in the same year. In addition to the lure of getting something for nothing, reasons for the illegal use of software include [1, 2]: no quality loss in the unauthorised digital copies; and widespread disregard of the fact that software is valuable intellectual property by users, despite their awareness that unauthorised use and duplication of software is illegal. There are further dimensions to illicit use of software that are

often overlooked. Firstly, if licensing compliance is a difficult process to implement, it is likely to be ignored. This effect can be seen with shareware that does not offer easy means of payment. Secondly, there may be mistaken infringement. In complex systems there can be a large number of components in use, and in such circumstances where there are multiple vendors, awareness and proper management of the many multi-faceted licensing agreements could be lacking even when users wish to comply.

There is considerable demand for methods that control access to, copying, and use of software products. Software vendors have long realised that they need to try to minimise software piracy. Some aspects of piracy can be mitigated by careful licensing arrangements, rather than improved security, especially if the licensing system has little or no overhead for the user. For lower value products, a complex licensing scheme may mean that the overheads of administering the licensing for the user will exceed the cost of the license for the product.

We describe here a model for comprehensive license management, specifically aimed at components and distributed systems. This model provides a module-loadable and policy-based mechanism that promotes adaptability and flexibility of any implementation. There have been some discussions of policy and frameworks in prior publications [10, 11, 12], which are extended in this paper. Section 2 of this paper covers related work since technical licensing first emerged, and sections 3 and 4 identify the challenges that are engendered by software license management and the differences between stand-alone packages and component software. The high level model is described together with discussion of its extensibility and flexibility in sections 5, 6 and 7, and the last section summarizes the benefits and potential drawbacks of this model. This paper does not address security and authentication issues that are necessary for any such system; the focus here is on a model for policies that will determine the licensing for components within

any such system. It is assumed throughout that details of authentication and security will be addressed in the future with a model at a larger scale of granularity than proposed here.

2. Related works

2.1. Activation keys and dongles

A number of technical licensing schemes have been developed over the years. They may be categorized as being based either on hardware or on software, although in practice a combination is often employed. The most common protection devices are hardware dongles and software activation keys, which are primarily used with shrink-wrapped, off-the-shelf software running on a single machine, or on smaller local area networks.

The hardware dongle is a hardware device that contains a key and must be connected to the computer in order for the application to execute, thus it may be seen as a licensing enforcement device – without the dongle the application will not execute. This kind of licensing control requires users to expend extra money on the hardware device, it has the inconvenience of requiring the user to attach extra hardware, it is difficult to upgrade when changes are essential, and it cannot be distributed electronically. There are also reports of compatibility problems in some configurations, particularly following operating system upgrades. Nevertheless, such padlock solutions often provide more protection than simple software locks. Hardware-based licensing management is still used in some places where flexibility is less important than security, though improvements with software protection devices, and perceived problems with peripherals when dongles are used, has led the vendors of some software, such as Autodesk's AutoCAD, to move away from dongle devices [13]. HASP of Aladdin Knowledge Systems [1], and the Sentinel of Rainbow Technologies [3] are typical in this kind of product. In contrast to dongles, software activation keys, where a special sequence of characters must be supplied during installation, exemplify simple methods that are widely used. Nowadays, most shrink-wrapped software products use activation key protection as a form of licensing enforcement. Though they may be cheaper and simpler to implement, software activation keys often provide less protection for the product than protection based on hardware, as the key with simple systems can be widely promulgated.

2.2. Traditional Technical License Management Service

As network technologies proliferate, license management services (LMS) [2] have emerged to adapt to the new environment. Most license management services implemented follow the client-server model. For the LMS, licensing information is encrypted, and the license server can only execute on an authorized host. Such a system is able to deal with several kinds of licenses, including concurrent, node-lock, demo, reserved, and shared licenses. Two typical LMS that provide these services are Macrovision's FLEXlm [4] and IBM's License Use Management (LUM) [5].

2.3. X-Open software license use management (XSLM)

The Open Group proposes an open standard called Software License Use Management (XSLM). The technical standard document specifies that the key factors driving the need for comprehensive license use management are the escalating software costs, the high administrative burden of license compliance control, the lack of effective customer control of software usage, and the lack of adequate protection for software publishers [6].

In today's computational environment, customers must deal with multiple products that are sourced from multiple software publishers, that can run on multiple platforms, and come with multiple licensing models. Given this exponential growth in complexity, the Open Group gives a list of requirements for an overall framework for license use management [6]:

- Extensible, flexible and comprehensive
- Independent of software publisher, platform, stand-alone or connected operations, and implementation
- Adaptable to future technologies
- Meet the needs of both customers and software publishers
- Cost effective for both customers and software publishers.

The Open Group also notes that license use management tools, processes, products, and systems must [6]:

- Provide facilities to encode license terms and conditions
- Record and report use level data
- Determine, report on, and verify compliance to license terms
- Allow customers to control and optimize the use of licenses within the terms and conditions of the license policy

- Allow software publishers to ensure their assets are protected.

IBM contributes to XSLM and developed the IBM License Manager (ILM) [7], which is based on the XSLM specification, as its licensing tool for zSeries Central Processor Complex (CPC). ILM is a combination of license management tools used to manage software licenses, check compliance with the software license, and manage the inventory of software licenses. It provides the technology that effectively enables implementing Workload License Charge (WLC) [7].

None of these license management tools provide answers for the movement in software development to component-based development in distributed environments, which is the focus of our license management model.

3. Software licensing challenges

To design a comprehensive license management model, we need to survey a variety of issues regarding software license management, including possible licensing methods, the license type, and the execution environment.

3.1. Technological License Management (TLM) and licensing contracts

Licensing services can be divided broadly into two types, those based on technology, and on contractual methods [8]. In practice, usually there is a combination of both. The license contract is a legal means of license management. It may not be as effective as technology-based license management (TLM). The TLM can provide evidence of the license contract and be a complement of the license contract. A good TLM is essential in today's fast-paced and complicated computational environment.

Initial models of dealing with software distribution relied on leasing or outright purchase of the package before use. Today, there are many means to facilitate software use whereby clients may try, lease, or buy the software. When clients choose to try the software before purchase, often it is provided as a demonstration, where clients can use a version of the software that offers limited functionality, or as an evaluation package that allows clients to make full use of the software for a limited period or for a limited number of times.

Clients, where the option is available, may choose the license arrangement, typically allocative or consumptive based. The main issues that face both the vendor and the client are the initial payment, the leasing period, and any monthly payments. There are many options for licensing. For example, a concurrent license defines a maximum number of usages at any one time; the site-locked license is only available for a single site; the floating license is available in a certain range of IP address; the usage-based license charges the client for every use; and the payment of the workload-based license depends on the workload of every access, such as how many CPU units are consumed. Clearly, given the complex usage environment, more and more sophisticated technical licensing tools are required.

3.2. Distributed Systems

Clients may use the software in a local or network-based environment. As improvements in network technologies proliferate, more and more software is run over a network rather than on a stand-alone, single machine. The networked and distributed systems environment provides licensing models with new challenges, often requiring multi-user and concurrent usage controls.

3.3. Software component licensing

Software reuse is much more than a populist mantra, offering major benefits for corporate and other large system users. There has been a rapid rise in component-based development over the last decade. The particular characteristics of component-based environments may require new licensing tools than those used in either traditional, stand-alone or simple networked environments.

Software components are undoubtedly different from shrink-wrapped or complete off-the-shelf packages. There are some questions that need to be answered when dealing with component systems. Does the licensing of a component differ from that of traditional software? If it does, what are the differences and how are they to be dealt with? The answer to these questions will help remove layers of obfuscation for system developers when facing licensing issues.

In [9] it is suggested that licensing models for software components are different from those for the traditional scenario, particularly with regard to the purpose, size, quality and flexibility. Indeed, in contrast to stand-alone software, the purpose of the software component is not to function independently. Software components need to be seamlessly integrated together to construct an application.

- High quality is assumed as a necessary requirement for a software component. Components are used to save time and money, and that is why they need to be protected with little overhead.
- The software component should afford adaptability to multiple platforms and future upgrades. At this time, detailed and

complete application programming interface (API) or source code is needed.

The legal terms in the software component agreements have a different slant as compared to terms used with earlier software, and may be more confusing.

- License Grant: The License Grant is the most critical provision in the license agreement, stipulating the rights the licensor grants to the licensee regarding the software component.
- Licensor or Licensee: Typically, the creator of a component will be a licensor when licensing the component to another, the licensee. However, these roles are often not clearcut (see below Transferring Roles).
- Target Objects/Components: The licensed target will not be a stand-alone or shrink-wrapped package, but rather a part of a software system.
- Grant: There are many kinds of grant (authorizations of types of use). Some of them are common, such as leasing, concurrent, floating, etc. There are also emerging types, such as workload-based. Others may be devised in the future.
- Transferring roles: As software components need to be integrated together to function correctly and usefully, there is often a change of status from licensee to that of both licensee and licensor (such as where the licensee creates a sublicense when the component is integrated into a system that is on-sold). This requires the transfer of some rights.
- Pricing and Payment: Based on different grant types, there are different pricing and payment options.

There are other terms and issues that specify aspects of the interaction and relationship between the licensor and the licensee after the purchase, such as:

- Liability: The liability of licensor may be limited to the direct damage caused by flaws or malfunction of licensed components. Misuse by the licensee may mitigate damages. This requires a mechanism to determine the extent of liability.
- Support and warranty: The licensor may provide support to legal users. Whether a particular use is authorized needs to be confirmed.
- Confidentiality: Sometimes the licensee needs to know the working mechanism or even hold the source code of the licensed

component. Non-disclosure agreements are common.

In short, for each component licensing questions involve various options, including payments, warranties and so forth, and lead to a complicated relationship between the licensor and licensee.

4. Policy-Driven Licensing Model

A flexible model is required because the licensing issues surrounding components are more complicated and diverse than those of stand-alone software, as discussed in the last section. A component may run in different situations under different licensing controls. We use policies to specify related licensing information, and interpret these policies at run time. A component may be associated with several licensing policies and may need to switch among these policies frequently. With a flexible policy driven mechanism, regardless of the complexity and mutability of the licensing issues, licensors can write or modify suitable policies for a component in a certain situation. The model also needs to be extensible, as the licensing system should be able to accommodate new requirements as they arise. A plug-in mechanism is a good solution to this requirement. By defining and publishing common interfaces, third party modules complying with these interfaces can be used. An out-of-date module can be replaced by a new one with little or no interruption to the system.

We have developed a policy-driven license management model that is both flexible and extensible for software components within an enterprise environment. The model is specifically oriented to component and distributed systems and to fit the software reuse paradigm. Based on this model, we can build a distributed software component that will provide the service of license management to other components. The model is illustrated in Figure 1. Three typical kinds of policies that are important to the system are shown: the Access Control (AC) Policy, the Billing (BL) Policy, and the Interface Filter (IF) Policy. When requested by a client, a licensed component automatically sends a license request to the license management component. This license request (LR) is a three-tuple:

LR = (S, T, P)

- S is the subject, the client who requests usage of the licensed component.
- T is the target/object, the licensed component being requested by S.
- P is the password or other identification of the end user S to use the licensed component.
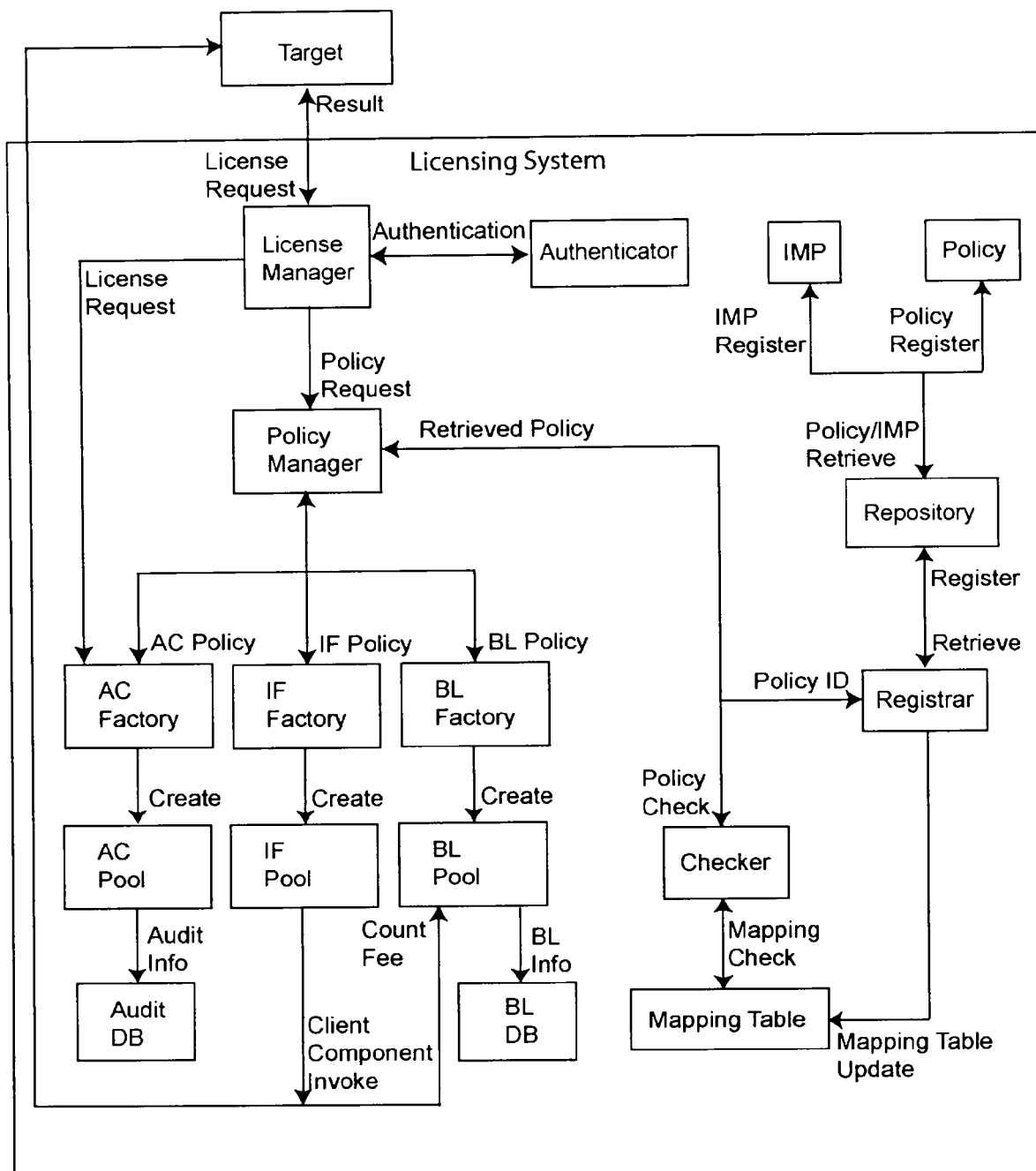
AC: Access Control
IF: Interface Filter
BL: Billing
DB: Database
IMP: Implementation Module / Building Block

Figure 1. The Conceptual Model

The license request is first sent to the **License Manager**. The **License Manager** separates the target, also called the object or the licensed component, from the licensing component, and works as a communication bridge between them. It cooperates with the **Authenticator** and the **Policy Manager**. The **License Manager** requests the **Authenticator** to authenticate this request (based on the information contained in the request). If the subject is not a legal user, the **Authenticator** sends negative feedback to the **License Manager** and the request is denied. Otherwise, the **Authenticator** identifies the requesting subject and associates it with a "role", that indicates a group of subjects with the same license type, that is, what they are able or unable to do when the licensing scheme is the same. Once a license request is authenticated, all the corresponding information regarding this license can be automatically retrieved from the repository by the licensing component based on the role and the license request. Normally no further identification of this subject or request is needed during the current session. The **License Manager** then sends this role along with its requested target to the **Policy Manager**. The **Policy Manager** is the core of this model, being responsible for most of the message handling and transmissions. Once the request is received, the **Policy Manager** first checks the **Mapping Table** that presents the mapping which is dependant on the role, and then targets a certain policy group. The structure of this mapping table is shown in Figure 2.

According to the target and role of the received request, corresponding policy group identity (ID) is acquired from the **Mapping Table**. Depending on this ID, the **Policy Manager** retrieves necessary license policies, including the access control policy, the interface filter policy and the billing policy, through the **Registrar**.

Target 1	Role 1	Policy Group 1.1
Target 1	Role 2	Policy Group 1.2
...	...	...
Target n	Role m	Policy Group n.m

Figure 2. Structure of Mapping Table

The license **Policy** is a set of rules widely used in our model to improve flexibility. There are several types of policies, including access control, billing, and interface filter policies. The details will be discussed in sections 6 and 7.

The interpretation of access control policies, interface filter policies and billing policies is implemented through the Policy Enforcement Pattern [14], although for better clarity they are addressed in three parts here. Other policy types that are necessary for licensors can be specified and interpreted using the same pattern. Applying the pattern it can be seen that the factory is responsible for invoking policy enforcement, creating the necessary component implementations, and putting them into the pool. Inside the pool, corresponding policy enforcement runs to enforce the respective policy.

After policy retrieval, the **Policy Manager** sends the access control policy to the **Access Control (AC) Factory**, which parses the access control policy, invokes the access control policy enforcement, creates instances of appropriate access control components (module implementation) and puts them into the **Access Control Pool**. For each role and target combination, there is a default access control policy in case the license management component has not defined any specific access rule. For example, one licensor requires that, for each target, there exists one "starter" role as default that only has partial read privilege to the target and is denied to all other privileges; or another licensor defines the "starter" role as no privilege at all. The **License Manager** next sends the license request to the **Access Control Pool** where the access control is realized, and if conditions of special events are satisfied, as determined by the licensors or the system, the request is audited based on the policy.

The **Access Control Pool** contains an aggregation of access control components that are running, access control policy decision parts, and a variety of policy enforcement parts. When a license request arrives, the policy decision checks the privilege of the subject on the target in the corresponding access control policy to determine whether or not to allow the request. If the subject has privilege on the target, the policy enforcement executes the access control policy. Depending on the different conditions of policies, different access control module implementations will be activated to carry out the policy. After this process has completed, an acknowledgement is sent back to the **Policy Manager**. Based on the returned acknowledgement, the **Policy Manager** decides whether or not to continue its ensuing work. Clearly, a negative acknowledgement of authorization will lead to denying the request and sending warning to the client. The request will also be audited.

If the request is authorized, the **Policy Manager** sends both the interface filter policy and the billing policy to their factories respectively.

There may be an interface filter policy for a certain request if the corresponding role is not granted full access to the target; that means some interfaces of the target will be filtered and their access will be prohibited while the target is invoked and running. The access control policy is analogous to the manager who only indicates what a role can and cannot do. Then the interface filter policy is like the worker who is responsible for how to prevent these unauthorized actions in practice. Both kinds of policy cooperate to realize the access control in this model. Similar to the process of the access control policy, the **Policy Manager** also sends the interface filter policy to the **Interface Filter (IF) Factory**, which builds up required filter enforcements and puts them into the **Interface Filter Pool**.

The **Interface Filter Pool** includes the active interface filter components, interface filter policies and the interface filter policy enforcements. The idea of delegation, first introduced by Lieberman [15] in the framework of a prototype-based object model, is used here.

Depending on the filter policy, there may be the need to dynamically prohibit a client's access to some declared interfaces of the target component, but normally we cannot touch the source code of the target. Here, delegation helps to achieve dynamic component adaptation. When necessary, a new component, which can understand and enforce filter policy results, is created at run-time. Allowed functionalities of the original target component are "re-wired,, into the new component and non-allowed ones are suppressed. The client interacts with the newly created component instead of the original one. If the client is accessing the legal interface, the request will be forwarded to the original target component; otherwise, appropriate process inside the new component will take action. The **Billing (BL) Factory** builds the billing policy enforcement and the corresponding billing components, depending on the policy, and puts them into the **Billing Pool**. The **Billing Pool** contains a variety of active billing components, as well as policy enforcement parts. We can provide a billing module (which can also be a component) to the target component that can use this module to gather useful license-related information and send it to the **Billing Pool**. The policy enforcement in this pool then analyzes the collected information and invokes in-pool components according to the billing policy to measure the license usage. The usage information is stored into the billing database for later reference.

If a technical solution for a licensing issue exists, it can be developed into a policy enforcement by utilizing the plug-in mechanism and by calling appropriate building blocks (components) once they have been specified in the configuration policy as discussed in section 5. This makes the license management component extensible so as to be able to satisfy future changes. If a new technical solution becomes available after the development of the licensing component, and the current policy enforcement is not compatible with it, a new solution can be implemented using policy rules, which gives great flexibility to the licensing component. The introduction of policy rules is addressed in section 6.

5. Extensible licensing framework

Plug-in Module Mechanism: Many license services in our component licensing model can be provided as predefined modules or components. They can be loaded at run-time. To fully take advantage of this mechanism, it is necessary to publish the specifications of common component interfaces, which enables a third party to join the implementation of licensing systems based on our policy-driven licensing model. By using this loadable module mechanism, our model is highly extensible and robust for future evolution. The loadable module mechanism can be applied to these specified parts:

- Policy Enforcement, including enforcement of access control policies, billing policies, interface filter policies, and other policies, if any
- Access Control Components
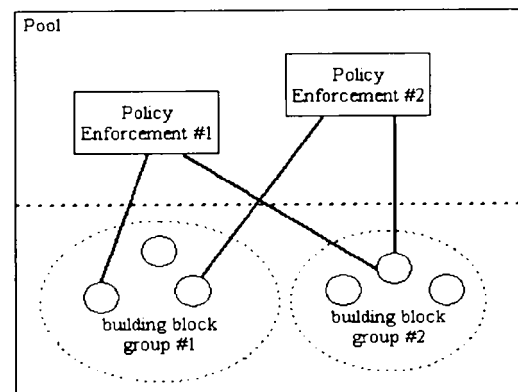- Billing Components
- Interface Filter Components

Figure 3. Loadable module mechanism

There are two kinds of loadable component. One is the policy enforcement; the other is the

implementation component, also called the building block. Their interactions are illustrated in Figure 3. The building blocks are many independently developed and delivered modules or components, which provide relatively independent and singular licensing functions. The policy enforcements are a variety of pre-defined procedures used to enforce routines that are well-known and regularly used in a licensing system. Each policy enforcement involves a series of actions, and each action may have several implementations. Those implementations can be independently built as building blocks and be integrated into the license management system as needed. During execution, policy enforcement needs to choose a suitable implementation from a group of available building blocks, as shown in Figure 3.

Normally the license management system will supply the policy enforcements and building blocks, however, based on a published specification, a third party can develop and complement these loadable components as well. As we may utilize many license modules and components in this component licensing model, two requirements emerge: First, how to select from the candidate components of a licensing functionality? Second, how to feed input to a selected component? Here we propose the configuration policy to solve these two problems.

Configuration Policy: This is a type of structural text file containing two types of license information. One is general license information, including the vendor identity, the product key, the license type and its properties. The other is the action selection that helps the policy enforcement decide how to choose a certain implementation from a building block group for its actions and provide input data for the implementation. It specifies which loadable license module or component should be loaded at run time, and what the parameters of this license component are. Figure 4 shows how the configuration policy works. The configuration policy is parsed and enforced by the appropriate policy enforcement part. Depending on the result of the policy parsing, the correct choice of action is made and corresponding modules are invoked to carry out the policy.

It is our policy-driven licensing model that provides the basis for the configuration policy. The configuration policy, in turn, is the key to licensing in component-based development software.
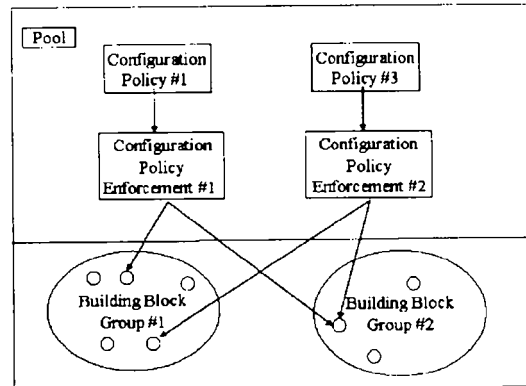

Figure 4. Configuration Policy Enforcement

A simple example of License Configuration Policy:

```
License (companyA, {1D6CDCA0-6916-11D2-A1C5-
0060081C43D9})
    {
        //General license information
        product = Banking_PayBill;
        product_version = 2.0;
            product_key = {FD57D29A-4FD9-11D1-
854E-00C04FC31FD3};
            //License type information and
properties
            license_key = {9AC04F56-957A-44EF-
BC87-00732D67FFC8};
            license_type = evaluation |
concurrent | float;
            expiration_date = 12/31/2002;
            concurrent_user = 5;
                allowed_IP_range  =
192.168.0.1-192.168.0.255;

            //Action selection for policy
enforcement
                    //Decryption the =    RSA;
encryption algorithm
            encryption_key_group = 71;//Supply
the parameter for RSA
    ......
    }
```

6. Licensing policy rule framework

If the pre-defined policy enforcements cannot satisfy a particular situation, we can use the policy rules to fulfill the requirement. The use of policy rules makes the license management services in the component licensing model very flexible. Similar to the configuration policy, the policy rules can be written by the licensors or developers and be registered into the policy repository for later use. Furthermore, the policy rule is able to combine several policy enforcements along with building blocks to form a new complex license management service. Figure 5 shows how it differs from the working mechanism of configuration policies.

A **Policy Rule** is an executable script file. It is an extension of the configuration policy, because,

besides containing the same kind of information as described in a configuration policy, it also contains some simple data processes and it can invoke certain license components depending on the results of these processes.
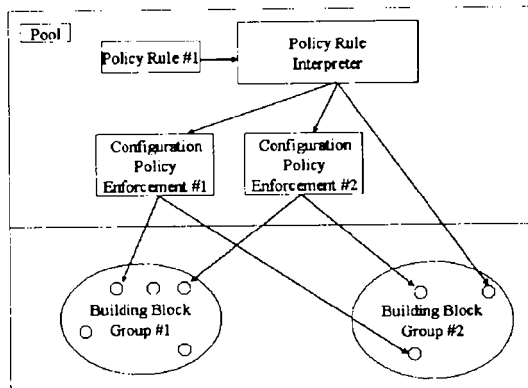

Figure 5. Policy Rule Interpretation

The policy rule is interpreted by the interpreter. Based on the interpretation result, the interpreter may invoke suitable configuration policy enforcement parts to parse the configuration information inside the policy rule. Meanwhile, as specified above, the policy rule contains its own action choice part that may be somewhat different from that in the configuration policy. The interpreter also calls the necessary modules directly. Thus, from the description above, we can see that the policy rule is more powerful and flexible than the configuration policy.

As mentioned earlier, not only can developers of the license system write policies according to the licensor's requirement, but licensors themselves can also write policies and register them into the policy repository. As the licensors are more familiar with their component products, they could take full advantage of this flexible tool, developing policy rules that best fit their licensing requirements. To use the script-based policy rule, our licensing model must fulfill several requirements:

Basis of scripting:
- The license management system that interacts with the scripted policy rule must be based on a good design, both component-based and object-oriented.
- The script-based interaction requires a stable architecture for extensible components.
- The license management system should embed an object-oriented script language.

To interpret the license policy at run-time:
- The Policy Manager retrieves corresponding access control policies, billing policies and

interface filter policies from the license policy repository and sends them to the respective factory.
- The factory parses the related policies. The license components required are extracted and the invocation information is used to create the instance in the pool.
- A pool contains policy enforcement parts that are responsible for the interpretation of respective policies.

An example of License Policy Rule:

```
License_Grant (companyA, {1D6CDCA0-6916-
11D2-A1C5-0060081C43D9})
{
        //General license information
        product = Banking_PayBill;
        product_version = 2.0;
        product_key = {FD57D29A-4FD9-11D1-854E-
00C04FC31FD3};
//License type information and properties
        license_key = {9AC04F56-957A-44EF-BC87-
00732D67FFC8};
        license_type = evaluation | concurrent
| float;
        expiration_date = 12/31/2002;
        concurrent_user = 5;
        a l l o w e d _ I P _ r a n g e       =
192.168.0.1-192.168.0.255;

        //Policy rules
        RSA_encryption_key_group = 71; //Supply
the parameter for RSA algorithm
        tripleDES_encryption_key_group  =  53;
//Supply the parameter for tripleDES

        if (is_licensetype(evaluation))
set_encryption(RSA,
RSA_encryption_key_group); //while handling
evaluation, use RSA
        else {
set_encryption(tripleDES,
tripleDES_encryption_key_group); //to all
other types, use tripleDES
        }
......
}
```

7. Discussion

We are currently implementing this model using both C++ and Java. Many aspects of the implementation are still under exploration. Even though policy-driven control has been widely used in Quality of Service (QoS) and proved practical, use of such a model in a component licensing environment is novel.

7.1. Benefits

- Flexibility: Our component licensing model allows licensors to define their own policies to meet specific requirements.
- Adaptability: The predefined license components in the repository can be independently developed and delivered.

This ensures that new functions and more optimized elements can be integrated into the system.

- Scalability: Our conceptual model fits different requirements, ranging from a small license system to an enterprise-scale license management system. For small and simple systems, the license components are relatively simple. For enterprise system, the simple license components may be combined to provide complex services, and more complicated license components may be integrated into the system when needed.

7.2. Drawbacks

- Performance: This component licensing model uses plain text as the policy description, which needs to be interpreted at run-time. This will slow down the policy execution. However, incorporation of more features for flexibility will mean more overhead.
- Comprehensibility: To implement this component licensing model is not a trivial task. It requires a sound system design, which in turn depends on detailed and thorough studies of the domain.

7.3. Performance issues and the solution

The use of the pools and pre-built license components provide partial solutions to the performance problem. Well-planned distribution and deployment also helps improve the response time.

- Load Balancing: The pool reduces overload of license component instantiation.
- Performance improvement by using pre-built modules: Pre-built license modules and components provide time-optimized modules that can accelerate the execution and improve performance dramatically.
- Distribution and deployment: Keep the licensing component at the same geographical location of the protected component. This will speed up the interaction between the components.

7.4. Future work

As a framework, our policy-driven licensing model is component platform neutral. To interact with different components, some adaptation to fit with popular component standards, such as CORBA, J2EE/EJB, and DCOM/COM+/.NET, is needed.

8. References

[1]: Aladdin Knowledge Systems, Inc., "Hardware against Software Piracy (HASP),,, 2002, http://www.aks.com/hasp, See also: http://www.siia.net/piracy/ and http://www.bsa.org/

[2]: Francisco Vilar Brasileiro, Trcio Rodrigues Bezerra, Walfredo Costa Cirne Filho, and J. Antao Beltrao Moura, "Bouncer: A Robust and Flexible License Management Service for Avoiding Illegal Use of Software,,, Proceedings of the XI Brazilian Symposium on Software Engineering, 1997

[3]: Rainbow Technologies, "Software Licensing and Protection: Sentinel™ Webpage,,, 2002 http://www.rainbow.com/sentinel

[4]: Macrovision & Globetrotter Software, "FLEXlm Overview Webpage,,, 2002 http://www.globetrotter.com/flexlm/lmover.shtml

[5]: International Business Machines Corporation, "IBM License Use Management White Paper,,, 1998 ftp://ftp.software.ibm.com/software/lum/misc/lumwhtp.pdf

[6] The Open Group, "Systems Management: Software License Use Management (XSLM),,, 1999 http://www.opengroup.org/onlinepubs/009619399/toc.htm

[7]: International Business Machines Corporation, "Planning for Workload License Charges,,, 2001 http://publibz.boulder.ibm.com/epubs/pdf/c0z2w102.pdf

[8]: Object Management Group, "CORBA Licensing Service Specification V1.0,,, 2000, http://www.omg.org

[9]: Andrea Chavez, Catherine Tornabene, and Gio Wiederhold, "Software Component Licensing: A Primer,,, IEEE Software, v.15 n.5, 1998

[10]: Ali Arsanjani, "Rule Object Pattern Language,,, Proceedings of PLoP2000, Technical Report #wucs-00-29, Dept. of Computer Science, Washington University Department of Computer Science, 2000

[11]: N. Damianou, N. Dulay, E. Lupu and M. Sloman, "The Ponder Policy Specification Language,,, Proceedings of the Policy Workshop 2001, HP Labs, Bristol, UK, Springer-Verlag, P.29-31, 2001

[12]: Benoit Dupire and Eduardo B Fernandez, "The Command Dispatcher Pattern,,, Proceedings of the 8th Pattern Language of Programs, Monticello, Illinois, 2001 http://jerry.cs.uiuc.edu/~plop/plop2001/

[13]: Autodesk AutoCAD 2002 "Questions and Answers,, http://usa.autodesk.com/adsk/files/ 669687_AutoCAD2002_QA_Standalone_Lic_Mgtl.pdf

[14]: Y. Zhou, Q. Zhao and M. Perry, "Policy Enforcement Pattern,,, Proceedings of the 9th Conference on Pattern Language of Programs 2002, Monticello, Illinois, 2002 http://jerry.cs.uiuc.edu/~plop/plop2002/final/ZZPerry_PLOP.pdf

[15]: Henry Lieberman, "Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems,,, Proceedings OOPSLA '86, ACM SIGPLAN Notices, 21(11):214–223, 1986 http://web.media.mit.edu/~lieber/Lieberary/OOP/Del egation/Delegation.html