Paweł Pietras[*], *Paweł Słowikowski*[*]

MOSTEK BEZPIECZEŃSTWA DLA PROTOKOŁU INTERNET INTER–ORB PROTOCOL

1. Wstęp

Rozwój standardu CORBA i powstanie protokołu IIOP pozwoliło na tworzenie rozproszonych aplikacji obiektowych w systemach heterogenicznych. Protokół IIOP umożliwił komunikację między obiektami brokerów żądań różnych producentów. Zaczął też być wykorzystywany jako protokół do komunikacji obiektów apletu w języku Java uruchamianego w środowisku przeglądarki WWW z obiektami działającymi na serwerze, zastępując mniej elastyczny i uniwersalny protokół CGI (*Common Gateway Interface*). Celowe stało się więc udostępnienie obiektów CORBA w sieci Internet. Udostępnianie tych obiektów wymaga jednak rozwiązania dwóch problemów.

Pierwszy problem dotyczy sieci chronionych ścianą ogniową. Niektóre z tych urządzeń umożliwiają komunikację przez jeden lub kilka portów TCP i UDP, co uniemożliwia bezpośrednie stosowanie protokołu IIOP. Rozwiązaniem tego problemu jest tunelowanie protokołu IIOP (*HTTP tunneling*) przez inne protokoły; jest to jednak podejście niestandardowe i niezalecane. Innym sposobem udostępnienia obiektu może być takie uruchomienie obiektu, aby jego adaptor obiektowy był uruchomiony na określonym porcie TCP otwartym przez ścianę ogniową. Można też uruchomić obiekt zastępczy (*proxy*) udostępnianego obiektu na porcie otwartym przez ścianę ogniową.

Drugi problem dotyczy zapewnienia kontroli dostępu do obiektów. W przypadku protokołu IIOP ochrona przez ścianę ogniową nie wystarcza, gdyż odbywa się na poziomie warstw niższych niż warstwa protokołu IIOP. Kontrola dostępu do obiektów CORBA powinna być zrealizowana przy użyciu pojęć standardów CORBA i IIOP. Wymaga to stworzenia systemu działającego na poziomie tych warstw, współpracującego ze ścianą ogniową.

Systemem takim może być mostek bezpieczeństwa. W przypadku jego współpracy ze ścianą ogniową możliwe jest zapewnienie przy spełnieniu kilku warunków pełnej kontroli dostępu do obiektów brokera. Umożliwia też pełną ochronę obiektów udostępnianych przez

* Katedra Informatyki, Akademia Górniczo-Hutnicza, Kraków

mechanizm tunelowania HTTP przy użyciu serwera *GateKeeper*. Kiedy nie jest dostępna ściana ogniową mostek może być pomocny w kontroli dostępu opartej na stworzeniu dwóch rozłącznych domen nazw wewnątrz brokera żądań. Nie zapewni się jednak w ten sposób pełnej kontroli dostępu, dopóki obiekt – klient będzie miał możliwość bezpośredniej komunikacji z chronionym obiektem na poziomie protokołu TCP.

Mostek może też być pomocny w rozwiązaniu pierwszego z opisanych problemów, jeśli umożliwia uruchamianie obiektów zastępczych (*proxy*) działających na określonych portach TCP.

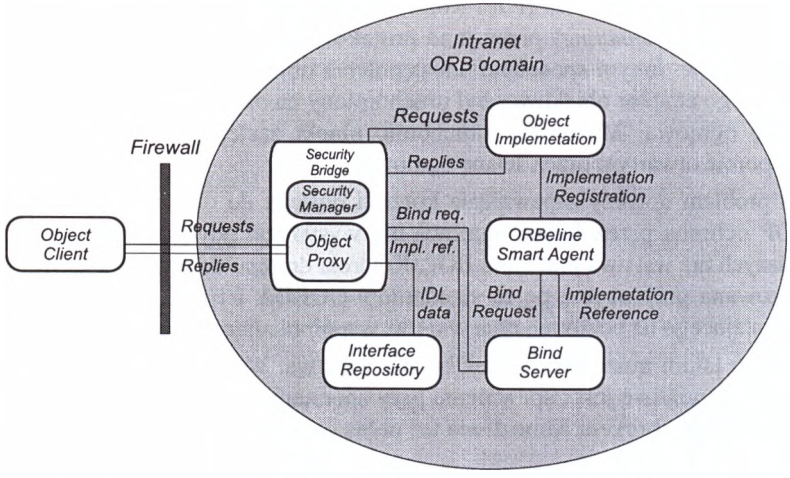
Mostek IIOP-IIOP z odpowiednio rozwiniętym mechanizmem kontroli dostępu umożliwiającym realizację zaawansowanego mechanizmu bezpieczeństwa przy korzystaniu z CORBA 2.0 i IIOP z zastosowaniem mechanizmu proxy został z powodzeniem zaimplementowany i przetestowany.

2. Broker żądań z mostkiem bezpieczeństwa

W proponowanym rozwiązaniu zastosowania mostka do komunikacji klienta z implementacją obiektu postanowiono wykorzystać mechanizm podobny jak w mostku *WonderWall* firmy IONA. Jego istotą jest stworzenie obiektu *proxy* działającego na określonym numerze portu udostępnianym przez ścianę ogniową. Możliwe jest rozwiązanie architektury brokera żądań z mostkiem bezpieczeństwa przy zastosowaniu jednej lub dwóch domen nazw (*name domain*). Istnieje też wariant architektury z wykorzystaniem dwóch domen nazw oraz GateKeepera.

2.1. Architektura brokera żądań z mostkiem bezpieczeństwa z jedną domeną nazw

Jest to najmniej skomplikowany sposób użycia mostka. Architektura brokera żądań przy zastosowaniu tego rozwiązania została przedstawiona na rysunku 1.



Rys. 1. Architektura brokera żądań z mostkiem bezpieczeństwa z jedną domeną nazw

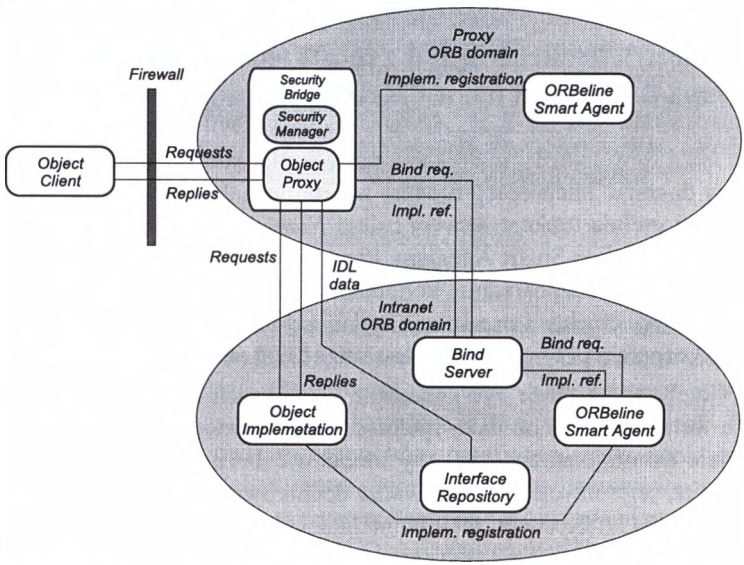
Klient (*Object Client*) korzysta z implementacji obiektu (*Object Implementation*) za pośrednictwem obiektu proxy działającego na określonym porcie TCP. Ściana ogniowa musi być skonfigurowana tak, aby umożliwiać komunikację TCP przez ten port. Jednocześnie, aby zapewnić maksymalne bezpieczeństwo, należy zablokować inne, nie używane porty. Obiekt *BindServer* służy mostkowi do uzyskiwania referencji proksyfikowanych obiektów. Byłby on w przypadku tej architektury brokera zbędny, lecz został wprowadzony w celu wykorzystania w architekturze przedstawionej w następnym podrozdziale.

Klient musi znać referencję IOR obiektu *proxy*. W przypadku zastosowania architektury z jedną domeną nazw istnieje możliwość takiego skonfigurowania ściany ogniowej, aby umożliwić komunikację klienta z ORBeline Smart Agentem poprzez określony port UDP i umożliwić mu korzystanie z *Name Service*, czyli uzyskiwanie referencji *proxy* poprzez normalne wywołanie bind (). Kryje się w tym jednak niebezpieczeństwo, gdyż klient może uzyskać referencję każdego obiektu zarejestrowanego w ORBeline Smart Agencie. Jeśli ściana ogniowa nie jest poprawnie skonfigurowana, może być możliwa komunikacja z obiektami z pominięciem *proxy*.

Za kontrolę dostępu do obiektów *proxy* odpowiedzialny jest obiekt mostka *Security-Manager*.

2.2. Architektura brokera żądań z mostkiem bezpieczeństwa z dwiema domenami nazw

Architektura ta różni się od architektury przedstawionej w poprzednim podrozdziale. System brokera żądań podzielony jest tu na dwie odrębne domeny nazw poprzez użycie dwóch niezależnych ORBeline Smart Agentów. Schemat współpracy obiektów został przedstawiony na rysunku 2.



Rys. 2. Architektura brokera żądań z mostkiem bezpieczeństwa z dwoma domenami nazw

Domena *Intranet ORB domain* jest zwykłą intranetową domeną nazw. Musi być w niej dodatkowo uruchomiony obiekt *BindServer*, który służy mostkowi do uzyskiwania referencji obiektów z tej domeny. Mostek widoczny jest w domenie *Proxy ORB domain*. Obiekty *proxy* rejestrowane są także w tej domenie. Klient może korzystać z usług implementacji obiektów za pośrednictwem *proxy*. Klient musi znać referencje IOR *proxy*. Można także, tak jak w przypadku architektury z jedną domeną nazw, umożliwić komunikację przez protokół UDP z ORBeline Smart Agentem z domeny *Proxy ORB domain*. Klient może jednak w ten sposób uzyskać wyłącznie referencje obiektów z tej domeny. Nie istnieje zatem ryzyko niekontrolowanego uzyskania referencji obiektów z domeny *Intranet ORB domain* pod warunkiem uniemożliwienia komunikacji przez protokół UDP z ORBeline Smart Agentem z tej domeny.

Kolejną zaletą takiej architektury jest możliwość zastosowania tej samej nazwy dla implementacji obiektu i jego proxy. Jest to możliwe wskutek rozdzielenia przestrzeni nazw. W takim przypadku klient używający metody bind () nie widzi różnicy między uzyskaniem referencji implementacji obiektu a uzyskaniem referencji jego *proxy*. Jedyna różnica polega na konieczności podania innego portu UDP podczas inicjalizowania brokera żądań poprzez parametr uruchomieniowy lub zmienną środowiskową OSAGENT_PORT.

Porównując architekturę brokera żądań przedstawioną w tym podrozdziale z architekturą z podrozdziału poprzedniego można stwierdzić, iż:

- jest ona bardziej elastyczna,
- gwarantuje większy stopień bezpieczeństwa,
- jest bardziej przezroczysta z punktu widzenia klienta.

Istotną zaletą rozwiązania z dwoma domenami jest możliwość jego wprowadzenia bez zmiany dotychczasowej architektury brokera żądań. Wymagane jest tylko uruchomienie obiektu *BindServer* i repozytorium interfejsów.

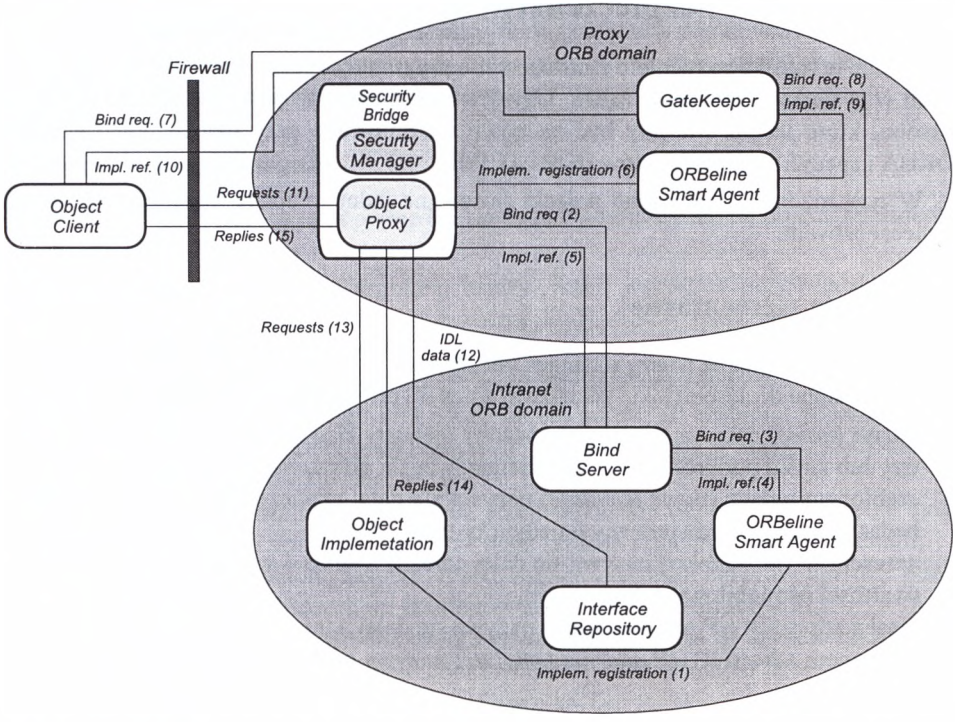
2.3. Architektura brokera żądań z mostkiem bezpieczeństwa z dwiema domenami nazw i GateKeeperem

W starszych niż 2.5. wersjach *VisiBroker for Java* aplet korzystający z usług obiektów VisiBrokera, aby dokonać inicjalizacji brokera żądań, musiał skontaktować się z GateKeeperem. *GateKeeper* spełnia funkcję serwera usługi *Name Service*, umożliwiając apletowi pośredni kontakt z ORBeline Smart Agentem. Poza tym, w przypadku istnienia ściany ogniowej i braku możliwości bezpośredniej komunikacji klienta z implementacją obiektu, *GateKeeper* tworzy *proxy* obiektu i przekazuje żądania do implementacji przesyłane pomiędzy apletem i GateKeeperem przy użyciu tunelowania HTTP (*HTTP tunneling*).

Użycie GateKeepera może być konieczne w starszych wersjach *VisiBroker for Java*. W nowszych wersjach może on dalej spełniać funkcje serwera *Name Service*. Jeśli ściana ogniowa będzie skonfigurowane tak, aby umożliwić bezpośrednią komunikację z *proxy* obiektu z mostka, jego użycie nie wprowadzi dodatkowych opóźnień w stosunku do rozwiązania bez użycia GateKeepera. Jeśli komunikacja będzie się odbywać przy użyciu tunelowania HTTP, należy liczyć się z dodatkowymi stratami efektywności. Niemniej możliwa jest kontrola i rejestracja dostępu do implementacji obiektów przez mostek.

GateKeeper może być zastosowany w obydwu opisanych poprzednio architekturach brokera żądań. Jednakże w wariancie z jedną domeną nazw istnieje możliwość dostępu do każdego obiektu z pominięciem kontroli przez mostek. W przypadku wariantu z dwoma domenami nazw niebezpieczeństwo to nie istnieje. Należy jednak pamiętać o takiej inicjalizacji GateKeepera, aby umożliwiał on dostęp do ORBeline Smart Agenta z domeny *Proxy ORB domain.*

Na rysunku 3 określono kolejność kroków wywołania metody implementacji obiektu przez mostek. Niektóre z nich są opcjonalne lub nie zawsze występują. W dalszej części opisano kroki wywołania.



Rys. 3. Architektura brokera żądań z mostkiem bezpieczeństwa z dwoma domenami nazw i GateKeeperem

Użytkownik uruchamia proces zawierający implementację obiektu w domenie intranetowej. Implementacja ta jest rejestrowana przez ORBeline Smart Agenta (krok 1). Zarządca mostka tworzy *proxy* danej implementacji. Proxy może być także automatycznie stworzone przez mostek wskutek mapowania referencji przekazywanych jako parametry żądań. Przy pomocy obiektu BindSrv mostek uzyskuje referencję obiektu reprezentowanego przez *proxy* (kroki 2 do 5) i rejestruje *proxy* w ORBeline Smart Agencie domeny internetowej (krok 6). Aplet klienta zwraca się do GateKeepera z domeny internetowej z żądaniem referencji obiektu i uzyskuje referencję *proxy* (kroki 7 do 10). Krok 1 występuje tylko na etapie uruchamiania implementacji obiektu, kroki 2 do 6 jedynie na etapie tworzenia *proxy*, a kroki 7 do 10 tylko na

etapie wiązania (*binding*) referencji obiektu w aplecie. Teraz następuje właściwe wywołanie metody. Klient przekazuje żądanie do mostka (krok 11). Zarządca bezpieczeństwa mostka sprawdza dopuszczalność wywołania i, w przypadku jego niedopuszczalności, sygnalizuje wyjątek klientowi. Jeśli wywołanie było poprawne, a w pamięci podręcznej mostka nie występuje definicja metody, jest ona pobierana z repozytorium interfejsów (krok 12, opcjonalny), a następnie żądanie przekazywane jest implementacji obiektu (krok 13). Po wykonaniu metod przez implementację odpowiedź zwracana jest do mostka (krok 14), a następnie do klienta (krok 15).

3. Zarządzanie bezpieczeństwem

Przy budowie mostka rozwinięto i zastosowano oryginalny sposób zarządzania bezpieczeństwem w systemach brokerów żądań. Użyto w nim intuicyjne pojęcia użytkownika, grupy i zasobu, które jednak musiały być na nowo zdefiniowane przy użyciu pojęć standardu CORBA i specyfikacji protokołów IIOP i TCP/IP.

W tym rozdziale omówiono pojęcia stosowane dalej w opisie koncepcji zarządzania bezpieczeństwem.

3.1. Pojęcia elementarne

Przez pojęcia elementarne należy rozumieć podstawowe pojęcia stosowane w opisie koncepcji, elementy dostępu do zasobu takie jak nazwa, adres IP, principal i odpowiednie szablony.

- nazwą (*name*) nazywa się dalej skończony niepusty ciąg znaków zaczynający się od litery lub znaku podkreślenia '_', zawierający znaki alfanumeryczne,
- szablonem nazwy (*name template*) nazywa się dalej skończony niepusty ciąg znaków będący nazwą lub zawierający pojedynczy znak '*',
- adresem IP (*IP address*) nazywa się dalej adres IP wersja 4 w postaci czterech oktetów (wartości ośmiobitowych bez znaku),
- maską adresu IP (*IP address mask*) nazywa się dalej adres IP,
- szablonem adresu IP (*IP address template*) nazywa się dalej parę (adres IP, maska adresu IP),
- principalem (*principal*) (zgodnie ze specyfikacją OMG CORBA 2.0) nazywa się dalej skończony, być może pusty ciąg oktetów (wartości ośmiobitowych bez znaku). Rola principala nie jest w specyfikacji jednoznacznie określona, jest on jednak w zasadzie przeznaczony do identyfikacji użytkownika,
- szablonem principala (*principal template*) nazywa się dalej wartość będącą principalem lub obiektem pustym (null).

3.2. Pojęcia złożone

Pojęcia złożone są to zdefiniowane w oparciu o pojęcia elementarne kluczowe pojęcia stosowane w koncepcji takie jak zasób, użytkownik, grupa i wywołanie i odpowiednie szablony.

- zasobem (*resource*) nazywa się dalej trójkę nazw reprezentujących odpowiednio określony interfejs, określony obiekt implementujący interfejs i operację tego interfejsu,

- szablonem zasobu (*resource template*) nazywa się dalej trójkę szablonów nazw mogącą sprecyzować interfejs, obiekt i operację bądź pozostawić dowolne z tych cech zasobu nieokreślone; w szczególności szablon zasobu może określać pojedynczy, a także dowolny zasób,
- użytkownikiem (*user*) nazywa się dalej parę (adres IP, principal) zawierającą określony adres IP oraz określony principal,
- szablonem użytkownika (*user template*) nazywa się dalej parę (szablon adresu IP, szablon principala) zawierającą szablon adresu IP mogący określać grupę adresów IP oraz szablon principala, mogący określać principal lub brak konieczności kontroli principala; w szczególności szablon użytkownika może określać pojedynczego, a także dowolnego użytkownika,
- grupą (*group*) nazywa się dalej skończony zbiór szablonów użytkownika,
- wywołaniem (*invocation*) nazywa się dalej parę zawierającą użytkownika i zasób, określającą konkretny dostęp do zasobu.

3.3. Relacje dopasowania

Relacje dopasowania są relacjami pomiędzy uprzednio zdefiniowanymi pojęciami złożonymi i szablonami.

- nazwa pasuje (*matches*) do szablonu nazwy, jeżeli wszystkie odpowiadające sobie znaki nazwy i szablonu nazwy mają równe kody ASCII bądź szablon nazwy jest ciągiem zawierającym pojedynczy znak '*',
- adres IP pasuje do szablonu adresu IP jeżeli dla każdego *i*-tego bitu każdego *j*-tego oktetu adresu IP a[j][i], *i*-tego bitu *j*-tego oktetu adresu IP szablonu adresu IP t[j][i] oraz i-tego bitu j-tego oktetu maski adresu IP szablonu adresu IP m[j][i] zachodzi relacja a[j][i] and m[j][i] = t[j][i] and m[j][i],
- principal pasuje do szablonu principala, jeżeli wszystkie odpowiadające sobie oktety principala i szablonu principala są równe bądź szablon principala ma wartość null,
- zasób R pasuje do szablonu zasobu Tr, jeśli nazwy interfejsu, obiektu i operacji zasobu R pasują do odpowiadających im szablonów nazw szablonu zasobów Tr,
- użytkownik U pasuje do szablonu użytkownika Tu, jeśli adres IP użytkownika U pasuje do szablonu adresu IP szablonu użytkownika Tu oraz principal użytkownika U pasuje do szablonu principala szablonu użytkownika Tu,
- użytkownik U należy do grupy G, jeśli istnieje szablon użytkownika Tu będący elementem grupy G taki, że U pasuje do Tu.

3.4. Szablon dostępu

Szablonem dostępu nazwano pojęcie koncepcji umożliwiające opis kontroli dostępu użytkownika do zasobu. Zdefiniowano relację pasowania wywołania do szablonu zasobu, będącą kluczową składową kontroli dostępu.

- szablonem dostępu (*access template*) nazywa się dalej relację, której dziedziną jest skończony zbiór grup, a przeciwdziedziną skończony zbiór szablonów zasobów,
- wywołanie I = (U, R) pasuje do szablonu dostępu A, jeżeli istnieje para (G, Rt) należąca do wykresu szablonu dostępu A taka, że U należy do G i R pasuje do Rt.

4. Kontrola dostępu

4.1. Mechanizm kontroli dostępu

Aby ułatwić administrowanie bezpieczeństwem, wprowadza się nazwy dla grup. Szablony zasobów wydają są wystarczająco dobrze i zwięźle opisane przez zawarte w nich szablony nazw interfejsu, obiektu i operacji, dlatego nie wprowadza się dla nich dodatkowych nazw.

Zarządzanie bezpieczeństwem w mostku związane jest z istnieniem czterech zbiorów opisujących zasoby, użytkowników oraz kontrolę dostępu użytkowników do zasobów. Należą do nich:

- zbiór grup (*group set*),
- zbiór zasobów (*resource set*),
- zbiór dostępów dopuszczonych (*allowed access set*),
- zbiór dostępów zabronionych (*denied access set*).

Zbiór grup jest skończonym zbiorem grup; opisuje on wszystkich użytkowników, którzy mogą uzyskać dostęp do zasobów poprzez mostek bezpieczeństwa. Nie może uzyskać dostępu poprzez mostek do żadnego zasobu użytkownik, który nie należy do żadnej z grup zbioru grup.

Zbiór zasobów jest skończonym zbiorem szablonów zasobów; opisuje on wszystkie zasoby, do których można uzyskać dostęp poprzez mostek bezpieczeństwa. Nie można poprzez mostek uzyskać dostępu do zasobu, który nie pasuje do żadnego z szablonów zasobów zbioru zasobów.

Zbiór dostępów dopuszczonych jest skończonym zbiorem, którego elementy należą do iloczynu kartezjańskiego zbioru grup i zbioru zasobów; jest to więc wykres szablonu dostępu zwanego szablonem dostępów dopuszczonych. Nie jest dopuszczalne wywołanie, które nie pasuje do szablonu dostępów dopuszczonych.

Zbiór dostępów zabronionych jest skończonym zbiorem, którego elementy należą do iloczynu kartezjańskiego zbioru grup i zbioru zasobów; jest to więc wykres szablonu dostępu zwanego szablonem dostępów zabronionych. Nie jest dopuszczalne wywołanie, które pasuje do szablonu dostępów zabronionych.

Kontrola dostępu do zasobu realizowana jest poprzez stwierdzenie dopuszczalności wywołania, nie musi być zatem kontrolowane otwieranie połączenia TCP/IP między klientem a serwerem. Nie jest oddzielnie sprawdzane, czy użytkownik z wywołania należy do jakiejś grupy ze zbioru grup, nie jest także sprawdzane, czy zasób z wywołania pasuje do jakiegoś szablonu ze zbioru zasobów. Zapewnione jest to przez warunek, że pary z obu zbiorów dostępów należą do iloczynu kartezjańskiego zbioru grup i zbioru zasobów. Zbiory grup i zbiory zasobów służą więc jedynie zarządzaniu mechanizmem bezpieczeństwa mostka.

Zgodnie z opisami zbiorów dostępów, aby wywołanie było dopuszczalne, musi pasować do szablonu dostępów dopuszczonych i nie może pasować do szablonu dostępów zabronionych. Zastosowanie dwu szablonów, jednego z selekcją pozytywną, a drugiego z negatywną jest celowe, aby było można ograniczyć dostęp przez mostek do niektórych określonych zasobów, umożliwiając dostęp do pozostałych.

4.2. Ocena koncepcji zarządzania bezpieczeństwem i mechanizmu kontroli dostępu

Zaproponowane podejście do kontroli dostępu do obiektów IIOP ma następujące cechy:

- ułatwia administrację kontrolą dostępu serwerów poprzez wprowadzenie pojęć grup i zasobów,
- nie wydaje się mieć istotnych wad w stosunku do podejścia opartego na listach dostępu, stosowanego w komercyjnych mostkach bezpieczeństwa,
- stworzenie efektywnej jego implementacji wydaje się być możliwe.

Ze względu na wymienione wyżej cechy postanowiono warstwę kontroli dostępu do zasobów w tworzonym pracy mostku oprzeć na podejściu przedstawionym w tym punkcie.

5. Założenia implementacyjne mostka

Podstawowym założeniem implementacyjnym była realizacja mechanizmu kontroli dostępu do obiektów CORBA IIOP opartego na podejściu zaprezentowanym w poprzednim rozdziale. Oto pozostałe istotne założenia implementacyjne, które były brane pod uwagę przy projektowaniu i implementacji mostka:

- Mostek udostępnia obiekty intranetowe poprzez tworzenie obiektów zastępczych (*proxy*) działających na określonym porcie TCP udostępnianym przez urządzenie ścianę ogniową.
- Kontrola dostępu do obiektów zastępczych jest realizowana według założeń z poprzedniego rozdziału.
- Konfiguracja kontroli dostępu powinna być wygodna, najlepiej oparta na GUI (*Graphical User Interface*).
- Zarządzanie obiektami zastępczymi powinno być również również wygodne i oparte na GUI.
- Powinno być możliwe zdalne zarządzanie bezpieczeństwem i obiektami zastępczymi poprzez sieć – najlepiej, aby GUI realizowany był przez odrębny proces komunikujący się z mostkiem poprzez protokół IIOP.
- Poza kontrolą dostępu do obiektów powinna istnieć także możliwość jego rejestracji (*logging*).
- Przy wszystkich powyższych założeniach mostek powinien być efektywny w sensie szybkości działania.
- W celu zmniejszenia czasu odpowiedzi mostek powinien być wielowątkowy.
- Pożądane byłoby, aby kod mostka był łatwo przenośny i działał zarówno w środowisku Solaris 2.X jak i Microsoft Windows 95/NT 4.0.

6. Wybór narzędzia implementacji mostka

Jako narzędzie i środowisko implementacji wybralino broker żądań VisiBroker for C++ 2.0. i VisiBroker for Java firmy Visigenic Software, Inc. Jest to implementacja brokera oficjalnie wspierana przez takie firmy jak Netscape i Oracle, o dużym stopniu zgodności ze specyfikacją CORBA 2.0. VisiBroker for C++ jest nową nazwą dobrze znanego ORBeline. Posiada on następujące cechy predestynujące go do realizacji mostka:

- jest dostępny w różnych środowiskach, w tym w Solaris 2.X i Windows 95/NT 4.0, używa (także wewnętrznie) protokołu IIOP,
- posiada działającą realizację DII/DSI wraz z repozytorium interfejsów,
- implementacje obiektów mogą być wielowątkowe,
- VisiBroker for Java umożliwia realizację GUI jako apletu w Javie możliwego do uruchomienia na dowolnej przeglądarce WWW z wbudowaną możliwością uruchamiania apletów Javy,
- posiada możliwość filtrowania otwierania połączenia z serwerem (*binding*) i wywołania operacji.

Cechy te w wystarczającym stopniu uzasadniają wybór VisiBrokera do realizacji mostka.

Kod źródłowy mostka został z powodzeniem skompilowany przy pomocy następujących kompilatorów:

- w środowisku Microsoft Windows 95/NT 4.0: Visual C++ wersja 4.2. z wykorzystaniem biblioteki Microsoft Foundation Classes wersja 4.2. firmy Microsoft Corporation,
- w środowisku Solaris 2.X: CC 3.0.1 firmy Sun Microsystems, Inc.

7. Strategie implementacyjne

7.1. Użycie repozytorium interfejsów

W mostku postanowiono użyć mechanizmu DII/DSI i repozytorium interfejsów do realizacji obiektu zastępczego (*proxy*). W stosunku do przekazywania komunikatów bez dekodowania ciała komunikatu (*message body*) rozwiązanie to ma następujące wady:

- wprowadza opóźnienia związane z kodowaniem (*marshaling*) i dekodowaniem (*unmarshaling*) komunikatu,
- wymaga obecności repozytorium interfejsów z załadowanymi definicjami odpowiednich interfejsów.

Zalety wynikające przyjęcia tego rozwiązania to:

- możliwość podmieniania referencji obiektów przekazywanych w parametrach i rezultatach na referencje obiektów zastępczych (*proxy*),
- możliwość dynamicznego tworzenia *proxy*,
- możliwość kontroli wartości parametrów (nie wykorzystywana),
- możliwość rejestracji wartości parametrów w logu.

7.2. Realizacja kontroli dostępu do zasobów

Kontrola dostępu do zasobów odbywa się poprzez procedury obsługi zdarzeń implementacji (*implementation event handlers*), będące niestandardowym mechanizmem udostępnianym przez *VisiBroker*. Wymaga to stworzenia obiektu dziedziczącego z klasy PMC_EXT: :ImplEventHandler z pokrytymi metodami wirtualnymi, wywoływanymi w momencie zaistnienia pewnego zdarzenia, np. otwarcia połączenia z klientem (metoda bind ()) bądź otrzymania komunikatu IIOP typu żądanie od klienta (metoda pre_method ()), oraz zarejestrowania go jako obiektu obsługującego zdarzenie poprzez wywołanie metody reg_glob_impl_handler () obiektu klasy PMC_EXT: :HandlerRegistry.

Oto fragment deklaracji klasy PMC_EXT: :ImplEventHandler:

```
class _PMCEXPORT ImplEventHandler // C++
{
public:
//...
virtual void bind(const ConnectionInfo&,
CORBA: :Principal_ptr,
        CORBA: :Object_ptr) {}
//...
virtual void pre_method(const ConnectionInfo&,
        CORBA: :Principal_ptr,
        const char * /* operation name*/,
        CORBA: :Object_ptr) {}
//...
};
```

Jak widać w deklaracji, wewnątrz obydwu metod dostępne są następujące informacje:

- struktura PMC_EXT: :ConnectionInfo, zawierająca między innymi nazwę hosta klienta oraz numer portu, na którym otwarte jest połączenia po stronie serwera,
- principal klienta,
- wskaźnik obiektu, na rzecz którego wywołana będzie metoda,
- nazwę operacji (w metodzie pre_method).

Ponieważ mając wskaźnik obiektu docelowego bez trudu można uzyskać nazwę interfejsu i nazwę tegoż obiektu, mamy w tym miejscu informację wystarczającą do zezwolenia lub zabronienia wykonania operacji obiektu zastępczego. Wyjątek sygnalizowany wewnątrz tych metod zwracany jest do klienta, w szczególności może być to standardowy wyjątek CORBA: :NO_PERMISSION, sygnalizujący klientowi brak praw do określonego zasobu.

Konkludując, wewnątrz metody pre_method () można w sposób elegancki zrealizować implementację podejścia kontroli dostępu do zasobów opisanego we wcześniejszej części artykułu.

7.3. Użycie pamięci podręcznej (cache)

Aby zapewnić efektywność pracy mostka, należało zastosować mechanizm pamięci podręcznej (*cache*), czyli lokalnego przechowywania często używanych informacji pochodzących z sieci bądź będących nie dającą się stablizować funkcją argumentów zmiennych w czasie.

Wyodrębniono następujące potencjalnie czasochłonne operacje systemowe, brokera żądań i kodu mostka:

- korzystanie z repozytorium interfejsów,
- uzyskiwanie adresu IP na podstawie nazwy hosta,
- kontrolę dopuszczalności wywołania.

Projekt mostka uwzględnia rozbudowany system pamięci podręcznych, przyspieszających współpracę z repozytorium interfejsów, mapowanie nazw hostów na adresy IP i kontrolę dopuszczalności dostępu do zasobu. Pamięć podręczna powinna być możliwie efektywna. Zastosowane w mostku pamięci podręczne oparte są na mechanizmie tablic mieszających (*hash tables*) z zastosowaniem algorytmu LRU (*Least Recently Used*) do usuwania obiektów w przypadku ich przepełnienia.

7.4. Synchronizacja wątków

Istotną rzeczą przy wielowątkowości implementacji mostka jest zapewnienie synchronizacji dostępu do obiektów dzielonych między wątkami. Ponieważ nie istnieje w C++ przenośna biblioteka wspomagająca programowanie wielowątkowe, synchronizację postanowiono oprzeć na klasie Lock (zamek).

Klasa Lock służy do przenośnej realizacji wzajemnego wyłączania typu mutex (*mutual exclusion*). Sposób jego realizacji enkapsulowany jest w implementacji tej klasy. Przeniesienie mostka w środowisko innego systemu operacyjnego (w którym oczywiście musi być dostępne C++ ze standardowymi bibliotekami i VisiBroker) będzie wymagało odpowiedniego przedefiniowania implementacji klasy Lock. W środowisku Solaris 2.X synchronizację oparto na obiekcie mutex_t z biblioteki wątków POSIX (pliki nagłówkowe synch.h i thread.h), a w środowisku Windows 95/NT 4.0. na klasie CCriticalSection z biblioteki Microsoft Foundation Classes.

W przypadku zajęcia zamka obiektu dzielonego przez wątek i zaistnienia nieprzewidzianego wyjątku w czasie wykonania przed zwolnieniem zamka mogłoby nastąpić zakleszczenie (*deadlock*) z powodu niemożności zwolnienia zamka. Aby uniknąć takich sytuacji, wprowadzono klasę SafeLock służącą do bezpiecznego zajmowania zamka.

Zamek podawany jest konstruktorowi obiektu SafeLock i zwalniany w jego destruktorze. Zapewnia to zwolnienie zamka w czasie zwijania stosu po zaistnieniu nie przechwytywanego wyjątku. Użycie klasy SafeLock zapewnia także zwalnianie zamka przy wyjściu z funkcji dowolną ścieżką sterowania bez konieczności jego zwalniania *explicite.*

7.5. Synchronizacja apletu konfiguracyjnego i obiektów mostka

Istotnym zagadnieniem w projekcie implementacji mostka jest synchronizacja wizualizowanej przez aplet konfiguracji obiektów mostka z rzeczywistym jej stanem. Można założyć, że kilku administratorów próbuje niezależnie konfigurować mostek, albo jeden administrator konfiguruje go przez dwa lub więcej uruchomionych jednocześnie apletów. Na pewno nie jest to częsty przypadek, ale należy założyć możliwość jego wystąpienia. Narzucające się rozwiązania to:

- zastosowanie wzajemnego wyłączania całego apletu.;
- mechanizm synchronizacji ze „śladem czasowym" (*time stamp*).

Wzajemne wyłączanie mogłoby dotyczyć całego apletu, to znaczy tylko jeden aplet mógłby konfigurować obiekty mostka. Jednak uniemożliwiałoby to jednoczesne konfigurowanie mostka z dwóch apletów. Wewnątrz implementacji mostka wzajemne wyłączanie dotyczy dostępu do obiektów takich jak zarządca *proxy* (mostek), zarządca bezpieczeństwa

i zarządca pamięci podręcznej. Ponieważ zdecydowano się na dopuszczenie możliwości jednoczesnej konfiguracji mostka z kilku apletów, zaprojektowano mechanizm synchronizacji danych przechowywanych w aplecie i stanu obiektów.

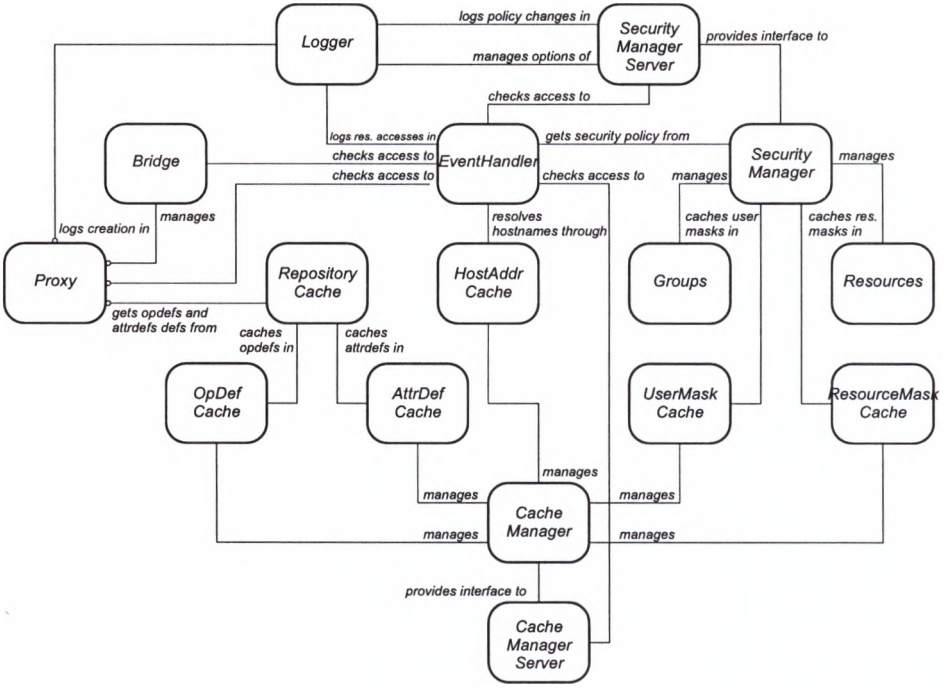
Każdy z obiektów konfigurowanych przez aplet posiada tak zwany „ślad czasowy" (*time stamp*). Ślad czasowy zmieniany jest przy każdej zmianie stanu obiektu, czy to z przyczyn zewnętrznych (w wyniku akcji użytkownika) czy wewnętrznych (na przykład automatyczne stworzenie obiektu *proxy* w wyniku mapowania przekazywanej referencji obiektu). Aplet z częstotliwością określoną przez użytkownika pobiera ślad czasowy i przy jego niezgodności odświeża stan obiektu w nim przechowywany i wizualizowany. Możliwa jest jednak próba dokonania zmian stanu obiektu przez aplet przed pobraniem jego aktualnego stanu. Dlatego każda funkcja mogąca dokonać zmiany stanu wymaga podania wartości przechowywanego w aplecie śladu czasowego. Jeśli wartość ta nie jest zgodna z wartością śladu w obiekcie, zwracany jest wyjątek CORBA: :BAD_INV_ORDER i aplet informuje użytkownika o fakcie rozsynchronizowania się danych apletu i stanu obiektu, proponując ponowną próbę dokonania zmiany. Istnieje też możliwość zwrócenia wyjątku CORBA: :PERSIST_STORE w przypadku, gdy obiekt przechowujący trwale swój stan napotkał błąd przy próbie zapisu do pliku.

8. Implementacja mostka

Uproszczony schemat mostka został przedstawiony na rysunku 4. Mostek składa się z następujących głównych obiektów:

- *Bridge*, czyli mostka. Mostek spełnia rolę obiektu zarządzającego obiektami Proxy, zawiera również obiekt RepositoryCache odpowiedzialny za pobieranie informacji z repozytorium interfejsów i przechowywanie jej w pamięciach podręcznych. Odpowiedzialny jest za zarządzanie obiektami Proxy z poziomu apletu konfiguracyjnego, synchronizację tego zarządzania i serializację informacji o obiektach Proxy.
- *SecurityManager*, zarządcy bezpieczeństwa. Obiekt ten jest odpowiedzialny za zarządzanie bezpieczeństwem i określanie dopuszczalności wywołań operacji oraz za serializację informacji o bezpieczeństwie. Zapewnia również serializację konfiguracji rejestratora.
- *EventHandler*, obiektu obsługującego zdarzenia. Otrzymuje on od bibliotek rdzenia brokera żądań informacje o otwieraniu i zamykaniu połączeń oraz o wywołaniach operacji. Zawiera obiekt HostAddrCache służący do mapowania nazw hostów na adresy IP i ich przechowywania w pamięci podręcznej. Odpowiedzialny jest za kontrolę dostępu. W celu określenia dopuszczalności wywołania używa obiektu SecurityManager.
- *CacheManager*, zarządcy pamięci podręcznej. Odpowiedzialny jest za zarządzanie systemem pamięci podręcznych i udostępnianie statystyk dotyczących dostępu i efektywności ich wykorzystania oraz serializację konfiguracji pamięci podręcznych.
- *Logger*, rejestratora. Rejestrator jest odpowiedzialny za rejestrację przez inne obiekty takich zdarzeń jak dostęp do zasobu, próba nieuprawnionego dostępu do zasobu, otwarcie i zamknięcie połączenia do implementacji obiektu, zmiana praw dostępu, tworzenia i usuwania obiektów Proxy oraz niektórych błędów. Przechowuje informacje o preferencjach dotyczących rejestracji.

- *SecurityManagerServer,* serwera zarządcy bezpieczeństwa. Jest to obiekt odpowiedzialny za konfigurację zarządcy bezpieczeństwa przez aplet konfiguracyjny za pośrednictwem IIOP. Zapewnia odpowiednią synchronizację dostępu do zarządcy bezpieczeństwa i zapis zmian konfiguracji do pliku we właściwym momencie. Służy również do konfiguracji z apletu rejestratora oraz umożliwia przeglądanie rejestrowanej informacji w aplecie.
- *CacheManagerServer*, serwera zarządcy pamięci podręcznej. Jest on odpowiednikiem obiektu SecurityManagerServer dla zarządcy pamięci podręcznej i ma podobny zakres funkcjonalności.



Rys. 4. Uproszczony schemat mostka

9. Zarządzanie mostkiem

Jednym z założeń było stworzenie narzędzia opartego na GUI i służącego do zarządzania mostkiem. Narzędziem takim jest aplet, zwany zarządcą mostka (*Bridge Manager*).

9.1. Założenia implementacyjne zarządcy mostka

1. Powinno być możliwe zdalne zarządzanie mostkiem poprzez sieć (również Internet):

- obiektami *proxy*,
- kontrolą dostępu do obiektów *proxy*,
- rejestracją dostępu do mostka oraz obiektów *proxy*.

2. Dostęp do mostka zdefiniowany jest poprzez interfejsy IDL. Dzięki temu zarządca mostka może być także używany do zarządzania innymi mostkami posiadającymi cechy funkcjonalności mostka przez nas zaprojektowanego oraz zaimplementowany identyczny interfejs IDL.
3. Komunikowanie się zarządcy mostka z mostkiem powinno odbywać się poprzez protokół IIOP.
4. Zarządzanie mostkiem powinno być wygodne, oparte na GUI.
5. Mostek może być równocześnie konfigurowany z wielu zarządców mostka.

9.2. Wybór narzędzia implementacji zarządcy mostka

Kod źródłowy został napisany w Javie i skompilowany w środowisku Microsoft Windows 95/NT: Visual J++ wersja 1.1. Zdecydowano się na Javę z następujących powodów:

- jest językiem obiektowym, o składni zapożyczonej z C++,
- posiada mechanizmy programowania zdarzeniowego i współbieżnego,
- dostarcza GUI,
- zapewnia przenośność programu na poziomie kodu binarnego, skompilowane klasy mogą być wykorzystywane na każdej platformie wyposażonej w Java VM (*Java Virtual Machine*), abstrakcyjną maszynę wirtualną. Obecnie dostępne są interpretery Java VM na platformy: Windows (Windows 95/NT), UNIX (Solaris, AIX, Linux), Macintosh (MacOS) i wiele innych. Umożliwia to wykonywanie skompilowanych programów napisanych w Javie praktycznie na każdym komputerze,
- pozwala na realizację programu jako apletu możliwego do uruchomienia na dowolnej przeglądarce WWW z wbudowaną możliwością uruchamiania apletów Javy (np. Netscape Communicator czy Internet Explorer),
- istnieje kompilator mapujący IDL na Javę.

Jako narzędzie i środowisko implementacji wybrano broker żądań VisiBroker for Java firmy Visigenic Software, Inc. Znakomicie spełnia on wymagania dotyczące zdalnego sterowania mostkiem ponieważ:

- używa protokołu IIOP,
- dostarcza GateKeepera,
- generuje na podstawie definicji interfejsów napisanych w IDL szkielety klas w Javie.

Istotnym powodem wyboru systemu VisiBroker for Java był oczywiście fakt implementacji samego mostka w brokerze VisiBroker for C++. Oba systemy zostały wyprodukowane przez jedną firmę i doskonale ze sobą współpracują.

10. Funkcjonalność zarządcy mostka

Z zarządcy mostka można korzystać po podaniu przeglądarce WWW adresu strony zawierającej aplet, zaraz po jego pomyślnym załadowaniu i uruchomieniu.

Zarządca mostka pozwala na konfigurowanie swoich parametrów działania takich jak czas odświeżania danych mostka oraz lista wyświetlanych komunikatów. Możliwość filtrowania wyświetlanych komunikatów przydatna staje się w przypadku krótkich czasów odświeżania, gdy istnieje możliwość pojawienia się ich w dużej ilość prawie jednocześnie.

Za pomocą zarządcy mostka można:

- Wybrać rodzaje zdarzeń zapisywanych przez rejestrator:
  - otwieranie i zamykanie połączeń,
  - dostęp do zasobów,
  - próby dostępu do nie udostępnionych zasobów,
  - błędy mostka czasu wykonania (np. brak definicji operacji w repozytorium interfejsów),
  - tworzenie i niszczenie obiektów *proxy*,
  - zmiany konfiguracji zarządcy bezpieczeństwa.
- Zarządzać obiektami *proxy*:
  - tworzyć proxy zarówno istniejące tylko do czasu zakończenia działania mostka jak i automatycznie tworzone po jego uruchomieniu,
  - pobierać tekstową postać referencji *proxy*,
  - zniszczyć *proxy*.
- Zarządzać grupami.
- Zarządzać zasobami.
- Nadawać prawa grupie.
- Ustalać prawa do zasobu.
- Ustalać parametry pamięci podręcznych mostka.

11. Skalowalność i wydajność

Istotnym zagadnieniem przy tworzeniu dowolnego systemu informatycznego, a zwłaszcza systemów sieciowych, jest ich skalowalność i wydajność. Wydajność mostka jest uzależniona od wydajności następujących składowych systemu:

- mechanizmu DII/DSI VisiBrokera,
- mechanizmu kontroli dostępu,
- sieci LAN.

Wydajność mechanizmu DII/DSI jest niezależna od implementatorów mostka. Komunikat IIOP musi być przez proxy mostka zdekodowany i ponownie zakodowany na podstawie informacji z repozytorium interfejsów. Informacja ta jest przechowywana w pamięci podręcznej, jednak pierwszy dostęp do niej może wprowadzić stosunkowo duże opóźnienie (*latency*). Konieczność zdekodowania i zakodowania parametrów i rezultatu operacji także wprowadza pewne koszty czasowe. Jednak bez zastosowania mechanizmu DII/DSI niemożliwe byłoby osiągnięcie automatycznego mapowania referencji obiektów.

Mechanizm kontroli dostępu także wprowadza opóźnienie w przekazywaniu żądań. Związane jest to z koniecznością mapowania nazwy hosta, przekazywanej procedurze obsługi zdarzeń przez broker żądań, na jego adres IP stosowany w mechanizmie kontroli dostępu oraz z czasem koniecznym na ustalenie prawa dostępu. Zastosowano mechanizm pamięci podręcznych w celu przyspieszenia mapowania nazwy hosta na adres i ustalania prawa użytkownika do zasobu.

W ramach pracy przeprowadzono testy w celu zbadania kosztów czasowych wnoszonych przez wszystkie składowe systemu mostka.

11.1. Ocena efektywności zarządcy bezpieczeństwa

Pomimo że przeprowadzona ocena efektywności opierała się na symulacji przeprowadzonej w warunkach różnych od warunków rzeczywistej sieci komputerowej, można wyciągnąć następujące wnioski:

- Algorytm sprawdzania dopuszczalności dostępu ma liniową złożoność w stosunku do ilości szablonów zasobów i użytkowników. Zostało to osiągnięte poprzez odpowiednią reprezentację danych zarządcy bezpieczeństwa (maski bitowe). Ocenia się, że przy bardzo dużej ilości szablonów zasobów lub bardzo licznych (w sensie ilości szablonów użytkowników) grupach złożoność ta będzie rzędu kwadratowego. Jednak przy ilości szablonów do ok. 2000 i grupach o ilości szablonów 4–32 nie zaobserwowano jeszcze nieliniowości.
- Przy odpowiednich rozmiarach pamięci podręcznej średni czas sprawdzania dostępu jest niezależny od ilości szablonów. W przeprowadzonej symulacji względny rozmiar wynosił 100%, ale w rzeczywistej sieci powinien być dużo niższy ze względu na tendencję do grupowania użytkowników i zasobów. Jest tak dzięki wykorzystaniu tablicy mieszającej (*hash table*) jako struktury implementującej pamięć podręczną.
- Przy braku pamięci podręcznej maksymalny średni czas kontroli dostępu wynosi 55 ms (dla ok. 2000 szablonów), co wydaje się być czasem możliwym do zaakceptowania. Czas taki (zależny od ilości szablonów) jest czasem opóźnienia (*latency*) dla wywołania, dla którego użytkownik i zasób nie znajdują się w pamięci podręcznej.

Administrator mostka powinien czuwać, aby współczynniki chybienia i wywoływania algorytmu LRU zarządcy bezpieczeństwa były możliwie małe. Aplet zarządcy mostka posiada wygodny interfejs do kontroli statystyk pamięci podręcznych. W przypadku dużej wartości tych współczynników należy zwiększyć rozmiary pamięci podręcznych i głów list. Rozmiar głów list powinien być liczbą pierwszą, najlepiej większą do połowy rozmiaru pamięci podręcznej. Należy pamiętać, że statystyki pamięci podręcznych są miarodajne dopiero po odpowiednio dużej ilości dostępów do tych pamięci, przynajmniej kilkukrotnie większej od ich rozmiaru.

Wnioski z testów efektywności zarządcy bezpieczeństwa świadczą o tym, że średnie opóźnienie (*latency*) wprowadzane przez warstwę kontroli dostępu jest małe i przy odpowiednim administrowaniu pamięciami podręcznymi zarządcy bezpieczeństwa nie powinno przekraczać 1 ms.

11.2. Ocena efektywności mostka

Testy kosztów czasowych związanych z warstwą fizyczną sieci LAN, mechanizmem DII/DSI i mechanizmem kontroli dostępu, a zatem z wszystkimi składowymi systemu, przeprowadzono w sieci LAN będącej fizycznie siecią Ethernet o przepustowości 10 Mb/s. Testy efektywności mostka wykazały, że w systemie Windows NT działającym na komputerze z procesorem Pentium z zegarem 75 MHz mostek wprowadza opóźnienie (*latency*) rzędu 10 milisekund niezależnie od rozmiaru komunikatu. Koszt czasowy kodowania i de-

kodowania 1 MB danych wynosi w takich samych warunkach ok. 1–1.8 sekundy. Należy również wziąć pod uwagę czas związany z dwukrotnym wydłużeniem drogi komunikatu w sieci LAN.

Ocena efektywności zależy od konkretnych zastosowań. Efektywność mostka jako internetowego systemu ściany ogniowej dla obiektów CORBA wydaje się bardzo dobra biorąc pod uwagę obecną przepustowość tej sieci.

12. Zakończenie

Opracowany mostek ma wiele zalet. Do najważniejszych można zaliczyć wygodną obsługę i możliwość zdalnej konfiguracji przy pomocy apletu zarządcy mostka oraz intuicyjne podejście do zagadnienia zarządzania bezpieczeństwem. Implementacja mostka ma takie cechy jak wielowątkowość, system pamięci podręcznych i mechanizm automatycznego odtwarzania zerwanych połączeń (*rebinding*), zapewniające dużą efektywność i stabilność działania. Kod źródłowy został opracowany z naciskiem na niezależność od platformy systemowej, co dało możliwość użytkowania mostka w systemach Solaris 2.X i Microsoft Windows 95/NT 4.0. oraz łatwą przenośność do innych systemów. Uzyskano to dzięki enkapsulacji funkcji systemowych związanych z siecią i synchronizacją wątków.

Łatwa przenośność kodu zarządcy bezpieczeństwa umożliwia jego wykorzystanie przy przenoszeniu mostka do innego systemu brokera żądań. Staranny projekt i implementacja w języku Java daje możliwość wykorzystania apletu zarządcy mostka w implementacji mostka dla innego brokera żądań praktycznie bez zmian.

Podstawowym zastosowaniem mostka jest zapewnienie bezpieczeństwa obiektom brokera żądań. Opracowany system umożliwia kontrolę dostępu do zasobów na podstawie adresu IP i principala klienta, rejestrację dostępów do zasobów oraz automatyczne tworzenie obiektów *proxy*.

Pomimo starań mostek ma jednak pewne ograniczenia – pracuje w systemie VisiBroker for C++ (wymaga posiadania tego oprogramowania i licencji na jego używanie), nie przekazuje wyjątków użytkownika i nie przekazuje kontekstu usługi obiektowej.

Pierwsze ograniczenie oznacza konieczność modyfikacji kodu mostka w przypadku konieczności jego wykorzystania w innym brokerze żądań. Usunięcie pozostałych dwóch ograniczeń będzie możliwe w przypadku rozszerzenia mechanizmu DII/DSI VisiBrokera lub przeniesienia systemu mostka do innego brokera żądań z mechanizmem DII/DSI w pełni zgodnym ze standardem CORBA 2.0.

Jakie są możliwości rozwoju mostka? Kod źródłowy mostka został napisany w sposób przenośny, dlatego jedną z nich jest jego przeniesienie pod inny system operacyjny, dla którego istnieje VisiBroker for C++. Istnieje też możliwość przeniesienia mostka do innego systemu brokera żądań, co może być jednak trudniejsze ze względu na prawdopodobnie inny sposób filtrowania żądań.

Można także w przyszłych wersjach mostka dodać możliwość kontroli dopuszczalności wywołania na podstawie innych parametrów połączenia i klienta, nie uwzględnionych obecnie. Wymagać to będzie jednak rozszerzenia koncepcji zarządzania bezpieczeństwem.

Literatura

[1] IONA Technologies Ltd.: *The Wonder Wall. CORBA IIOP Firewall Proxy, Version 1.0.* beta. 1997

[2] IONA Technologies Ltd.: *IIOP on the Internet (Firewall Navigation and WonderWall(tm)).* White Paper, 1997

[3] OMG: *The Common Object Request Broker: Architecture and Specification, Revision 2.0.* OMG TC document, 1995-1996

[4] OMG: *CORBA 2.0/Interoperability.* Universal Networked Objects. OMG TC document, 1995

[5] Visigenic Software Inc.: *VISIGENIC VisiBroker for C++ Programmer's Guide, Version 2.0.* 1996

[6] Visigenic Software Inc.: *VISIGENIC VisiBroker for C++ Reference Guide, Version 2.0.* 1996

[7] Visigenic Software Inc.: *VISIGENIC VisiBroker for Java Programmer's Guide, Version 1.0.* 1996

[8] Visigenic Software Inc.: *VISIGENIC VisiBroker for Java Reference Guide, Version 1.0.* 1996

[9] Mecnarowski L.: *Monitorowanie protokołu IIOP.* Praca magisterska, Katedra Informatyki AGH, Kraków 1996

[10] Pietras P., Słowikowski P.: *Mostek Bezpieczeństwa IIOP.* Praca magisterska, Katedra Informatyki AGH, Kraków 1997

Recenzent
prof. dr hab. inż. Krzysztof Zieliński