ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1999
Seria: INFORMATYKA z.36 Nr kol. 1414

Marek ŚLESICKI, Henryk KRAWCZYK
Politechnika Gdańska, Wydział Elektroniki, Telekomunikacji i Informatyki

METODY ZABEZPIECZANIA APLIKACJI SIECIOWYCH JAVA

Streszczenie. Przedstawiono charakterystykę aplikacji rozproszonych implementowanych w Javie, ze szczególnym uwzględnieniem mechanizmów zabezpieczających. Przeanalizowano aplety, serwlety i aglety i wskazano zasady bezpieczeństwa wykorzystywane w konstrukcji aplikacji. Podkreślono rolę zarządcy bezpieczeństwa jak i narzędzi wspomagających zapewnienie wysokiej wiarygodności funkcjonowania aplikacji.

METHODS OF SECURITY ASSURANCE FOR JAVA NETWORK APPLICATIONS

Summary. Java-oriented user applications are discussed and security models for applets, servlets and aglets are presented. Their suitability for improving security services are given. The paper describes also how to organize dependability processing in distributed information - processing systems.

1. Wprowadzenie

Dynamiczny rozwój sieci komputerowych, a także aplikacji rozproszonych sprawił, że oprócz problemów wydajności bardzo istotne stały się również problemy bezpieczeństwa [1]. Przez bezpieczeństwo systemu rozumie się na ogół [10]:

- autoryzowany dostęp użytkowników do zasobów systemu,
- zapewnienie poufności informacji poprzez odpowiednie jej kodowanie i umożliwienie dekodowania tylko przez jej właściciela,
- zapewnienie autentyczności polegającej na potwierdzaniu, że informacja rzeczywiście pochodzi od nadawcy, który się za niego podaje,

- zapewnienie integralności, co oznacza niemożliwość modyfikacji wiadomości w czasie przesyłania.

Zapewnienie całkowitego bezpieczeństwa systemu w praktyce nie jest możliwe. Dlatego też rozważa się różne poziomy bezpieczeństwa, których dobrym przykładem jest standard opisany w tzw. „Pomarańczowej Księdze", lub jego modyfikacje dla komputerowych systemów przetwarzania, tzw. kryteriów ITSEC (Information Technology Security Evaluation Criteria)[6], gdzie wyróżnia się 7 poziomów bezpieczeństwa :

E0 niedostateczne zabezpieczenie, wymagane jest przekonstruowanie systemu,

E1 nieformalny opis architektury zabezpieczeń,

E2 nieformalny opis koncepcji szczegółowej (kontrola konfiguracji i nadzór dystrybucji),

E3 szczegółowa koncepcja zabezpieczeń i jej implementacja w postaci kodu źródłowego,

E4 formalny model polityki zabezpieczeń : opis koncepcji ogólnej i szczegółowej,

E5 odwzorowanie formalnej koncepcji szczegółowej w kod źródłowy,

E6 formalny opis architektury zabezpieczeń zgodny z modelem formalnym polityki ochrony.

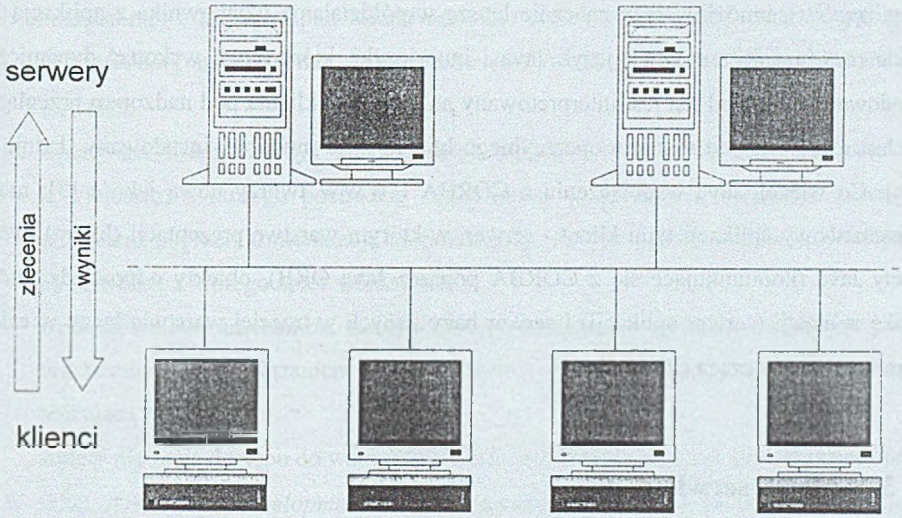
Niniejsza praca skupia się w głównej mierze na problematyce zapewnienia bezpieczeństwa w systemach rozproszonych opartych na technologiach web'owych [2]. Ograniczymy się do przedstawienia mechanizmów zabezpieczeń poziomu E3 dla typowego środowiska przetwarzania rozproszonego. Zajmiemy się również analizą przydatności rozwiązań konstrukcyjnych opartych na Javie, które mogą być wykorzystane do budowy bezpiecznych rozproszonych systemów web'owych. Analiza ta obejmuje : aplety, serwlety i aglety. Przeanalizaowano charakterystyki funkcjonalne i dotyczące zabezpieczeń, pozwalające na określenie możliwości wykorzystania różnego typu rozwiązań do budowy aplikacji rozproszonych z uwzględnieniem konkretnych wymagań bezpieczeństwa.

2. Przetwarzanie sieciowe klient – serwer

Przetwarzanie rozproszone realizowane jest najczęściej w architekturze klient - serwer, która rozdziela funkcje pomiędzy dwa typy niezależnych i autonomicznych procesów - - klienta i serwera. Klientem jest jakikolwiek proces wymagający obsługi ze strony procesu

serwera, którego zadaniem jest obsługiwanie żądań klientów. Zarówno klient, jak i serwer mogą rezydować na tym samym komputerze lub na różnych komputerach przyłączonych do sieci.

Architektura klient - serwer jest przeznaczona do pracy niezależnej od platformy sprzętowej, systemowej i lokalizacji aplikacji. Klient - serwer wymaga, aby stacja robocza dysponowała własnym środowiskiem systemowym oraz oprogramowaniem aplikacyjnym (klientem), umożliwiającym dostęp do serwera bazy danych. Serwer natomiast musi być wyposażony w oprogramowanie zarządzające bazą danych, czyli wykonujące wszelkie operacje na tym zbiorze. Powoduje to rozłożenie obciążenia z przechwyceniem czasochłonnej obsługi użytkownika na jego lokalną stację roboczą, przez co serwer bazy danych może wykonywać funkcje nieodłącznie związane z tą bazą.



Rys. 1. Architektura klient – serwer
Fig. 1. The client – server architecture

Pojęcie architektury klient - serwer może dotyczyć zarówno fizycznej architektury sprzętowej (patrz rys.1), jak też może się odnosić do oprogramowania aplikacyjnego. Aplikacja klient - serwer to zbiór połączonych w logiczną i funkcjonalną całość procesów, które są umieszczone na komputerach połączonych w sieć komputerową. Procesy te dzielimy na usługobiorców (klient) i usługodawców (serwer). Przyjęło się mówić, że komputer, na którym jest umieszczony proces klienta, to klient, zaś komputer z działającym procesem

serwera to serwer, jednak nic nie stoi na przeszkodzie, aby na jednym komputerze pracowały oba rodzaje procesów.

Omawiając aplikacje typu klient - serwer nie sposób nie wspomnieć o największych tego typu aplikacjach, skupiających ogromną ilość serwerów i jeszcze większą ilość klientów. Są to oczywiście aplikacje internetowe, takie jak WWW czy Gopher.

Pierwsza generacja aplikacji WWW to statyczne dokumenty hipertekstowe, które są ładowane i wyświetlane na uniwersalnym kliencie - przeglądarce, która może wyświetlać różne rodzaje dokumentów (tekst, grafika, dźwięk, video). Następnie, dzięki protokołowi CGI pojawiła się możliwość opracowywania odpowiedzi na serwerze. Oba podejścia zakładają, że dokumenty mogą być tylko przeglądane bez zapewnienia jakiegokolwiek bezpieczeństwa.

Następna generacja aplikacji WWW to aktywny klient z ładowalnymi komponentami (ang. *applets*), umożliwiający znacznie lepsze współdziałanie użytkownika z aplikacją [7]. Takie rozwiązanie umożliwił język Java i inne języki, które mogą wykonać dynamicznie załadowany kod. Kod ten jest interpretowany na maszynie klienta pod nadzorem przeglądarki. Jest niezależny od systemu operacyjnego klienta i nie musi być instalowany (ładuje się sam). Co więcej, Java w połączeniu z CORBA i WWW tworzy nową jakość [3]: model trójwarstwowy aplikacji typu klient - serwer, w którym warstwę prezentacji (klient) tworzą aplety Java (komunikujące się z CORBA poprzez Java ORB), obiekty odpowiedzialne za logikę aplikacji (serwer aplikacji) i serwer bazy danych w trzeciej warstwie łączy w całość warstwa pośrednicząca CORBA.

3. Aplety i serwlety

Java i aplety zrewolucjonizowały Internet, zaś „zawartość wykonywalna" stała się powszechnym terminem w słowniku każdego użytkownika Internetu. Zasadniczo, aplety postrzegane są jako kod programu, który może być załadowywany z serwera za pomocą sieci i natychmiast wykonywany na maszynie klienta, np. przez różnego rodzaju przeglądarki. Poza apletami często spotykamy się z serwletami. W tym jednak przypadku wykonanie kodu serwletu ma miejsce nie na maszynie klienta, lecz na serwerze, na którym ów serwlet został wcześniej zainstalowany. Dopuszczalne jest również, aby oprogramowanie klienta przesyłało kod programu do serwera.

Ze względów bezpieczeństwa zbiór funkcji dostępnych dla apletów jest znacznie ograniczony. Do realizacji swoich zadań aplety mogą korzystać z połączeń JDBC (*Java Database Connectivity*) oraz z rozproszonych obiektów. Przypomnijmy, że aplety muszą spełniać szereg zasad bezpieczeństwa, których spełnienie jest kontrolowane przez zarządców bezpieczeństwa przeglądarek. Najważniejsze zasady to [4]:

- Aplety nie mogą ładować bibliotek lub definiować podprogramów (metod native). Mogą używać tylko swego własnego kodu Java oraz elementów Java API dostarczonych przez przeglądarkę. Każda przeglądarka apletów musi udostępnić Java API zdefiniowane w pakietach Java.
- Aplety nie mogą w zwykły sposób czytać i zapisywać plików na komputerze, na którym są uruchomione. Aplety w przeglądarce mogą czytać pliki wyspecyfikowane przez URL (*Uniform Resource Locators*), zamiast nazwy pliku. Aby zapisać dane na komputerze, z którego aplet załadowano (serwer), można wysłać dane do aplikacji uruchomionej na serwerze, która może swobodnie czytać i zapisywać dane na serwerze.
- Aplety nie mogą tworzyć bezpośrednich połączeń sieciowych poza połączeniami z serwerem, z którego pochodzą. Aplet może się jednak kontaktować z innymi serwerami poprzez komunikację za pomocą np. aplikacji pośredniczącej, działającej na serwerze, z którego aplet załadowano.
- Aplet nie może uruchamiać żadnych programów na komputerze, na którym został uruchomiony. Rozwiązaniem tego problemu może być współpraca z aplikacją pracującą na serwerze.
- Aplety nie mają dostępu do wszystkich właściwości systemu (ang. *system properties*).

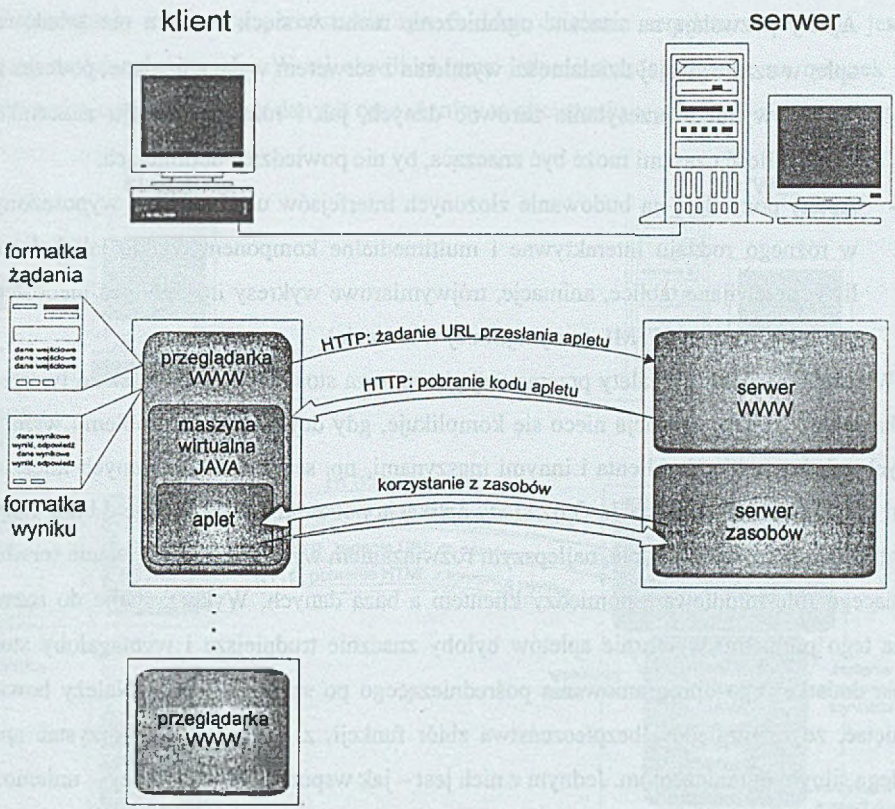
W skład JDK (*Java Development Kit*), popularnego pakietu do tworzenia wszelkiego rodzaju aplikacji Javy, a więc i apletów, wchodzi narzędzie o nazwie *javakey*, które jest używane do podpisywania plików JAR (*Java Archive*). Plik JAR jest archiwum tworzonym przez narzędzia wchodzące w skład pakietu JDK i może zawierać klasy Javy oraz inne dane, takie jak grafika czy też dźwięk. Przeglądarka apletów pozwala uruchomić z pełnymi prawami (tak jak aplikację lokalną) każdy aplet umieszczony w pliku JAR podpisanym przez zaufaną jednostkę (producenta oprogramowania, organizację certyfikującą itp.). Tak więc aplet taki nie podlega przedstawionym powyżej restrykcjom modelu bezpieczeństwa Javy.

Należy zwrócić uwagę na różnice między klasycznym modelem bezpieczeństwa Javy opartym na zarządcy bezpieczeństwa oraz przedstawionym powyżej mechanizmem opartym na mechanizmie podpisów. Chodzi tu głównie o różnicę między modelem kontroli dostępu, pozwalającym na kontrolę „drobnoziarnistą", tj. na poziomie elementarnych operacji dokonywanych przez aplet, a „binarnym" modelem zaufania występującym w przypadku apletów podpisywanych, który umożliwia kontrolę dostępu typu „wszystko albo nic". Najnowsze wersje Javy, a więc i zarządcy bezpieczeństwa, pozwalają podpisanym apletom również na selektywny dostęp do zasobów. Podjęcie decyzji o tym, z jakich zasobów aplet taki może korzystać, a z jakich nie może, należy do użytkownika.

3.1. Zalety korzystania z apletów

Graficzny interfejs użytkownika tworzony za pomocą apletów Javy jest bardziej elastyczny od typowego, wręcz ascetycznego interfejsu tworzonego za pomocą HTML'a (*Hyper Text Markup Language*), pozwala na tworzenie dynamicznych i łatwo modyfikowalnych przez użytkownika aplikacji. Wydaje się, że korzystanie z apletów Javy jest doskonałym sposobem tworzenia aplikacji klienckich, mimo że wymagane jest posiadanie do tego celu przeglądarki sieciowej wyposażonej w maszynę wirtualną Javy.

Na rys. 2 przedstawiono ideę działania apletów dla realizacji przetwarzania rozproszonego. W przedstawionej konfiguracji zadaniem apletu jest opracowanie danych wynikowych, które w postaci formatki wynikowej są przedstawiane użytkownikowi. Zazwyczaj do opracowania wyniku wykorzystywane są dane przekazane do apletu z formatki wejściowej oraz dane, jakie aplet zaczerpnie z serwera zasobów. Przypominamy, że mechanizmy bezpieczeństwa Javy narzucają fizyczną lokalizację serwera zasobów, z których może korzystać aplet, a mianowicie serwer ów musi być umieszczony na maszynie, z której pochodzi aplet.



Rys.2. Mechanizm działania apletów
Fig.2. Applets activity mechanism

Najważniejsze zalety apletów Javy w stosunku do HTML'a to:

- Aplety umożliwiają lokalną weryfikację wprowadzanych danych (co prawda istnieje również możliwość dokonywania takiej weryfikacji przy użyciu kombinacji HTML'a z JavaScript, jednak różnice w implementacji JavaScript'u przez różne przeglądarki powodują, że jest to zadanie dosyć trudne).
- Aplet może w sposób dynamiczny tworzyć, modyfikować i usuwać grupy i listy elementów i dopuszczalnych wartości określonych parametrów oraz zmieniać sposoby weryfikacji danych (HTML, nawet w połączeniu z JavaScript, nie pozwala na tego typu zmiany bez odwoływania się do CGI lub serwletów i budowania nowej strony HTML).

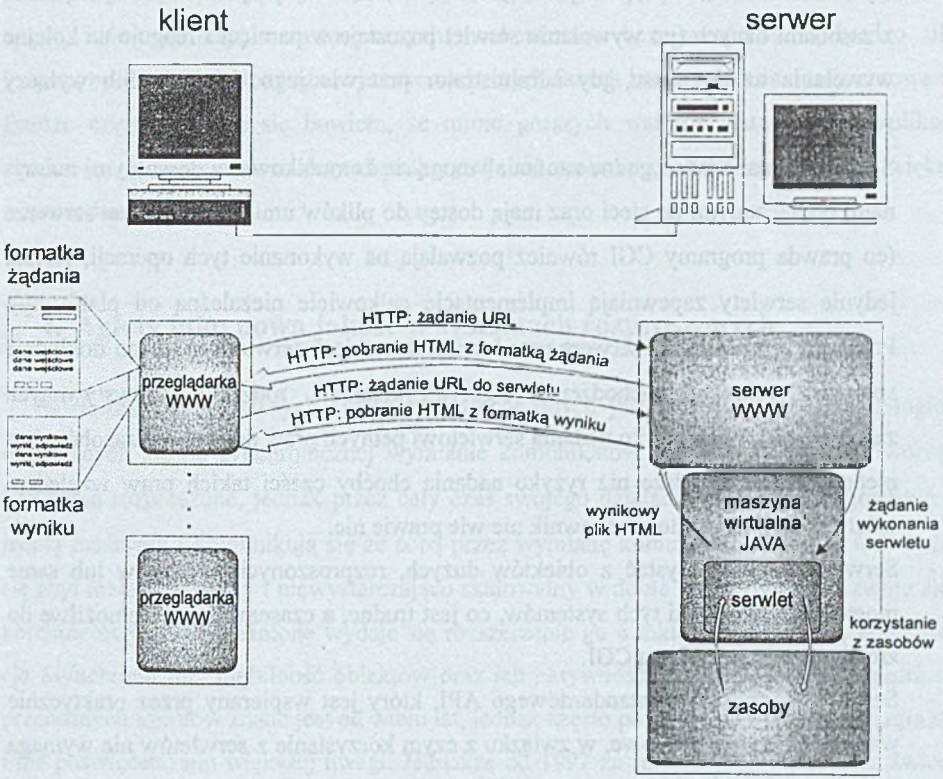
- Aplety pozwalają na znaczne ograniczenie ruchu w sieci, bowiem raz załadowany aplet w czasie swojej działalności wymienia z serwerem wyłącznie dane, podczas gdy HTML wymaga przesyłania zarówno danych, jak i różnego rodzaju znaczników, których ilość czasami może być znacząca, by nie powiedzieć dominująca.
- Aplety pozwalają na budowanie złożonych interfejsów użytkownika, wyposażonych w różnego rodzaju interaktywne i multimedialne komponenty, takie jak kolorowe listy, przewijane tablice, animacje, trójwymiarowe wykresy itp., które są niedostępne z poziomu języka HTML (przynajmniej w na razie).

Wymienione powyżej zalety przemawiają istotnie za stosowaniem apletów do tworzenia aplikacji web'owych. Sytuacja nieco się komplikuje, gdy dojdziemy do problemu wymiany danych między aplikacją klienta i innymi maszynami, np. serwerami baz danych mieszczącymi się na różnych maszynach. Otóż kiedy aplikacja potrzebuje połączyć się z bazą danych rezydującą na zdalnej maszynie, najlepszym rozwiązaniem wydaje się zastosowanie serwletu, pełniącego rolę middleware pomiędzy klientem a bazą danych. Wykorzystanie do rozwiązania tego problemu wyłącznie apletów byłoby znacznie trudniejsze i wymagałoby stosowania dodatkowego oprogramowania pośredniczącego po stronie serwera. Należy bowiem pamiętać, że ze względów bezpieczeństwa zbiór funkcji, z których może korzystać aplet, podlega silnym ograniczeniom. Jednym z nich jest – jak wspomniano wcześniej – uniemożliwienie nawiązania jakichkolwiek połączeń sieciowych z wyjątkiem połączeń z maszyną, z której pochodzi aplet. Istnieje co prawda możliwość skorzystania z tzw. sygnowanych apletów (apletów opatrzonych podpisem cyfrowym np. twórcy apletu), wobec których nie egzekwuje się ograniczeń w dostępie do zasobów, jednak procedura uzyskiwania i weryfikowania podpisów jest w chwili obecnej na tyle trudna i uciążliwa, że do rozwiązania tego ucieka się jedynie w skrajnych przypadkach. Należy tu jeszcze wspomnieć o odwiecznym problemie kompatybilności przeglądarek różnych producentów, których oprogramowanie w mniejszym lub większym stopniu odbiega od obowiązujących standardów.

3.2. Serwlety jako alternatywa apletów

Zadaniem serwletów jest realizacja zadań przekazywanych im przez przeglądarki sieciowe pracujące na maszynach klientów. Oprogramowanie WEB serwera przy użyciu maszyny wirtualnej Javy uruchamia serwlet [3]. Serwlet pobiera dane wejściowe ze strony HTML, przetwarza je, po czym zwraca wynik w postaci wygenerowanej nowej strony HTML. Ideę

funkcjonowania serwletów przedstawia rys. 3. Idea działania serwletu omówiona jest w dalszej części opracowania. W tej chwili chcemy jedynie zwrócić uwagę na rysunek pod kątem rodzaju wymienianych danych oraz różnicy w obciążeniu maszyny serwera i klienta.



Rys. 3. Mechanizm działania serwletów
Fig. 3. Servlets activity mechanism

Ze względu na podobieństwo działania serwlety są czasem utożsamiane z programami CGI (ang. *Common Gateway Interface*). Zarówno serwlety, jak i CGI są uruchamiane na serwerze i przetwarzają dane pochodzące ze stron HTML. Z tego też względu ich dalszy rozwój jest w znacznym stopniu ograniczony możliwościami języka HTML i skryptów Javy. Jednakże mimo znacznego podobieństwa serwlety przewyższają programy CGI pod wieloma względami, wśród których najważniejsze to:

- Wykonują się na WEB serwerze jako wątki, co jest znacznie mniej kosztowne niż w przypadku programów CGI, gdzie każde wywołanie CGI tworzy nowy proces.

- Szybkość wykonywania serwletu jest nawet kilkadziesiąt razy większa niż CGI realizującego to samo zadanie.
- Serwlety mogą przechowywać dane między kolejnymi wywołaniami, przy każdym wywołaniu nie muszą się inicjalizować i są w stanie utrzymywać trwałe połączenia z zasobami danych (po wywołaniu serwlet pozostaje w pamięci i reaguje na kolejne wywołania aż do czasu, gdy administrator przerwie jego działanie lub wyłączy serwer).
- Serwlety uznane jako „godne zaufania" mogą się komunikować z dowolnymi maszynami podłączonymi do sieci oraz mają dostęp do plików umieszczonych na serwerze (co prawda programy CGI również pozwalają na wykonanie tych operacji, jednak jedynie serwlety zapewniają implementację całkowicie niezależną od platformy). Ponieważ administrator serwera przed zainstalowaniem serwletu może go dokładnie sprawdzić (określić pochodzenie, algorytm działania, rodzaj wykorzystywanych zasobów itd.), więc ryzyko nadania serwletowi pełnych praw dostępu do zasobów jest nieporównanie mniejsze niż ryzyko nadania choćby części takich praw względem apletu, o którym często użytkownik nie wie prawie nic.
- Serwlety mogą korzystać z obiektów dużych, rozproszonych systemów lub same mogą być elementami tych systemów, co jest trudne, a czasem nawet niemożliwe do zrealizowania za pomocą CGI.
- Serwlety korzystają ze standardowego API, który jest wspierany przez praktycznie wszystkie serwery sieciowe, w związku z czym korzystanie z serwletów nie wymaga dodatkowych zabiegów ze strony administratora.
- Serwlety są stosunkowe łatwe do zaprojektowania i implementacji, zaś źle napisany lub uszkodzony w czasie eksploatacji serwlet nie powoduje zawieszenia pracy serwera.
- Serwlety Javy niosą ze sobą korzyści wypływające z samej Javy – pojawiające się problemy na wszystkich platformach są takie same.

Oba przedstawione rozwiązania posiadają zalety i wady, jednak wyboru między nimi dokonuje się uwzględniając potrzeby i zadania konkretnej aplikacji sieciowej. Co prawda, charakteryzują się one nieco odmiennymi własnościami i możliwościami, jednak bardzo często o wyborze po prostu decyduje złożoność budowanej aplikacji. Jeżeli posiada ona złożony i rozbudowany interfejs użytkownika, przez co przepływ danych jest znaczny, wtedy

zwykle ucieka się do rozwiązań opartych na apletach. I odwrotnie, jeżeli przepływ danych jest stosunkowo niewielki, najlepszym rozwiązaniem wydaje się skorzystanie z serwletów. Należy mieć również na uwadze sposób działania aplikacji oraz charakter i rodzaj zasobów, z których korzysta. Mówimy o tym z tego względu, że często ograniczenia nakładane na aplety, które wynikają z zastosowanych mechanizmów bezpieczeństwa są bardzo silne i muszą być dogłębnie rozważane już we wczesnych fazach konstruowania oprogramowania. Bardzo często okazuje się bowiem, że mimo gorszych walorów estetycznych aplikacje zbudowane w oparciu o serwlety charakteryzują się znacznie prostszą konstrukcją i większą niezawodnością.

4. Aglety jako nowa jakość w systemach rozproszonych

Większość współczesnych systemów rozproszonych jest tworzonych w technologiach opierających się na synchronicznej wymianie komunikatów. Co więcej, obiekty tworzące system są rozproszone, jednak przez cały czas swojego działania pozostają stacjonarnie na jednej maszynie i komunikują się ze sobą przez wymianę komunikatów. Model ten wydaje się zbyt mało elastyczny i niewystarczająco skalowalny w dobie intensywnego rozwoju sieci komputerowych. Uzasadnione wydaje się rozszerzenie go o takie własności, jak komunikacja asynchroniczna, mobilność obiektów oraz ich aktywność i autonomia [9]. Zagadnienie przenośnych agentów znane jest od wielu lat, jednak często postrzegane było jako marginalne i nie poświęcano mu większej uwagi. Jednakże od 1997 za sprawą IBM zaczęto na świecie mówić o agletach, technologii przenośnych agentów zbudowanych w oparciu o język Java. Mobilni agenci tworzą uniwersalny i kompletny model nowoczesnego przetwarzania rozproszonego, łącząc w sobie mechanizmy komunikacji synchronicznej i asynchronicznej, przesyłanie komunikatów i przesyłanie obiektów, obiekty stacjonarne z obiektami mobilnymi. Mając na uwadze fakt, że może istnieć wiele agletów realizujących wspólne zadanie, można z dużą dozą obiektywizmu stwierdzić, że mobilni agenci są przyszłością systemów rozproszonych [11].

Stosowane przez maszynę wirtualną Javy mechanizmy bezpieczeństwa są na tyle skuteczne, że pozwalają na bezpieczną i niemalże nieograniczoną „wędrówkę" agentów. Najbardziej zaawansowanym narzędziem do tworzenia aplikacji mobilnych agentów opartych na Javie jest pakiet IBM Aglet Workbench. Pakiet ten definiuje interfejs funkcji JAAPI (*Java*

Aglet Application Programming Interface) oraz dostarcza zbioru użytecznych narzędzi oraz przykładów ułatwiających zrozumienie zasad tworzenia agletów.

Aglety są autonomicznymi, dynamicznymi obiektami Javy mogącymi przenosić się z maszyny na maszynę za pomocą sieci [5]. Wykonujący się aglet może wstrzymać swoje wykonanie, przesłać się na inną maszynę i tam wznowić wykonanie od momentu, w którym zostało przerwane. Tak więc przenoszony jest zarówno kod agletu, jak i jego stan (tj. stan liczników, zmiennych, rejestrów itd.). Aglety mogą odbierać żądania zewnętrzne, jednak czy i jak będzie się dany aglet stosował do tych żądań, jest „jego wyłączną sprawą". Autonomia agletów wynika ze sposobu ich działania, w ogólności bowiem raz uruchomiony aglet samodzielnie decyduje, w jaki sposób i na jakiej maszynie będzie kontynuował swoje wykonanie. Wymienione własności agletów implikują ich kolejną zaletę, a mianowicie odpowiednio zaprojektowane potrafią znacząco zmniejszyć ruch w sieci oraz w sytuacjach awaryjnych kontynuować swoje działanie (w zależności od rodzaju działalności może być ono mniej lub bardziej skuteczne) nawet bez korzystania z sieci. W najbardziej niekorzystnej sytuacji mogą one oczekiwać na przywrócenie operatywności sieci.

4.1. Aglety i aplety

Aglety Javy rozszerzają model przenośnego za pomocą sieci kodu, którego popularność zawdzięcza się apletom Javy. Jako ciekawostkę można tu dodać, że słowo *aglet* powstało od słów *agent* oraz *applet*, co samo przez się podkreśla silny związek między agletami i apletami. Tak jak w przypadku apletów, pliki zawierające klasy agletów mogą „wędrować" poprzez sieć. Jednakże, jak wspomniano wcześniej, aglety w odróżnieniu od apletów potrafią przenosić również swój stan. Co więcej, aplet może być przesyłany jedynie z serwera do klienta. Aglet, jako działający program Javy (jak wiemy, program w fazie wykonanie opisywany jest zarówno poprzez swój kod, jak i bieżący stan), może wędrować między dowolnymi maszynami. Nic nie stoi na przeszkodzie, aby dany aglet wielokrotnie zmieniał maszynę kończąc swoją „wędrówkę" na maszynie, na której został uruchomiony, lub też na maszynie, na której jest oczekiwany.

Podobnie jak aplet, aglet jest wątkiem (lub kilkoma wątkami) działającym w kontekście lokalnej aplikacji Javy. Aplikacja taka jest tworzona przez przeglądarkę, bowiem służy ona jako „środowisko" dla apletów, które będą uruchamiane przez przeglądarkę. Aplikacja instaluje zarządcę bezpieczeństwa (ang. *security manager*), którego zadaniem jest egzekwowanie

restrykcji dotyczących pewnych działań apletu. Restrykcje te są stosowane w odniesieniu do apletów nie obdarzonych zaufaniem. Zaufanie takie można uzyskać stosując mechanizm podpisów, kiedy to mamy do czynienia z podpisanymi apletami (ang. *signed applets*). Aby załadować aplet, aplikacja tworzy klasę ładującą (ang. *class loader*), która wykorzystuje protokół HTTP do pobrania apletu z serwera.

Analogicznie, do przesłania i uruchomienia agletu wymagana jest również lokalna aplikacja Javy. Tak jak poprzednio, jest instalowany zarządca bezpieczeństwa, który ogranicza zbiór możliwych działań agletów nie obdarzonych zaufaniem. Zaufanie, niezbędne do przeprowadzenia pewnych operacji, można uzyskać stosując mechanizm podpisów. Aglet jest załadowywany poprzez klasę ładującą aplikacji, która korzysta np. z protokołu ATP (ang. *Agent Transfer Protocol*) w celu pobrania ze zdalnej maszyny kodu agletu i danych opisujących jego stan. Do działania agletu wymagane jest otoczenie oferujące interfejs funkcji JAAPI (ang. *Java Aglet Application Programming Interface*).

4.2. Bezpieczeństwo agletów

Problem bezpieczeństwa jest jednym z najczęściej poruszanych tematów przez użytkowników mobilnych agentów. Z jednej strony, są one wspaniałym środkiem pozwalającym na rozwiązanie wielu problemów świata rzeczywistego, z drugiej jednak strony, mogą posłużyć jako narzędzie do tworzenia różnego rodzaju niebezpiecznego oprogramowania, takiego jak wirusy. Wiele osób twierdzi, że zezwolenie na działanie na własnej maszynie programom nieznanego pochodzenia, które „przyszły sobie" przez sieć, jest otwartym zaproszeniem dla różnego rodzaju zagrożeń. Oprogramowanie takie jak *Aglets Framework*, które jest stosowane na maszynach „goszczących" aglety, implementuje rozbudowany wielowarstwowy model bezpieczeństwa. Pierwsza warstwa tego modelu pochodzi bezpośrednio od języka Java. Zadaniem tej warstwy jest wieloetapowe sprawdzanie kodu agletu, począwszy od sprawdzania poprawności kodu, na weryfikacji spójności kodu bajtowego Javy kończąc. Kolejną warstwą jest wspomniany już zarządca bezpieczeństwa, który pozwala użytkownikom oprogramowania typu *Aglets Workshop* na stosowanie własnych mechanizmów bezpieczeństwa w odniesieniu do agletów uruchamianych na jego maszynie. Inicjalnie, oprogramowanie to jest skonfigurowane w sposób bardzo restrykcyjny. Wykonanie przez aglet jakiejkolwiek operacji, która potencjalnie może być niebezpieczna dla systemu (np. dostęp do pliku), jest przez zarządcę bezpieczeństwa automatycznie blokowane. Ostatnią, trzecią

warstwą jest interfejs funkcji Java Security API, który umożliwia twórcom agletów łatwe i wygodne korzystanie z nowoczesnych mechanizmów bezpieczeństwa. Interfejs ten obejmuje wszelkiego rodzaju funkcje kryptograficzne, pozwalające na szyfrowanie, tworzenie elektronicznych podpisów, na autoryzacji kończąc.

W celu zapewnienia bezpieczeństwa podczas wykonania agletów system zarządzający musi być wyposażony w następujący mechanizmy:

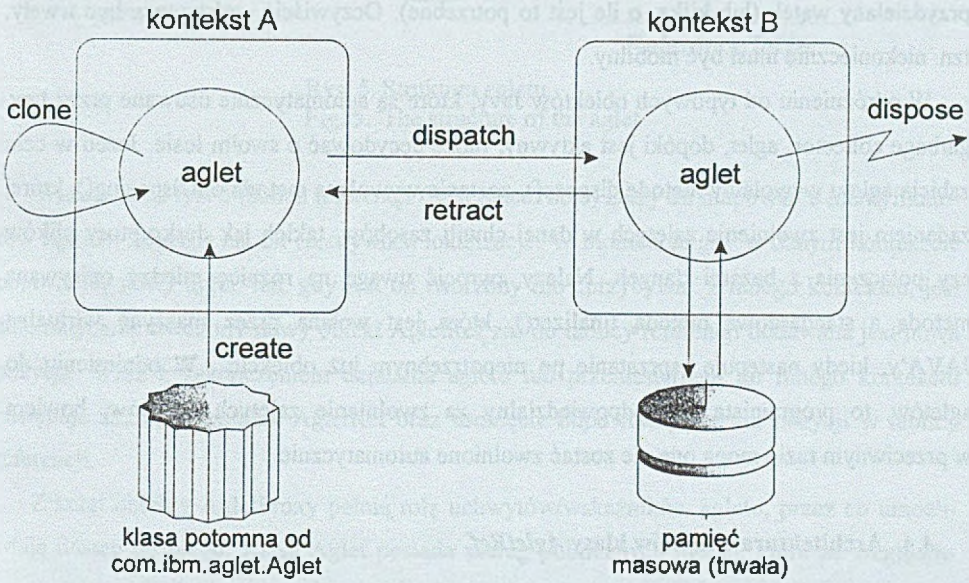
- identyfikacja i weryfikacja autentyczności nadawcy, producenta i właściciela agletu,
- autoryzacja agenta (lub jego właściciela) poprzez określenie zakresu stawianych mu restrykcji,
- bezpieczna komunikacja z innymi systemami w celu zapewnienia bezpieczeństwa agentowi,
- kontrola i monitorowanie poczynań agletu.

W systemach wspierających mobilnych agentów agenci muszą mieć jasno określoną tożsamość w taki sposób, aby mogły być kontrolowane oraz mogły być im udostępniane zasoby, w zależności od tego skąd pochodzą, kto jest ich twórcą itp. Z tego też względu bardzo ważną sprawą jest określenie w sposób niepodważalny zarówno twórcy, jak i użytkownika agletu. Wbrew pozorom jest to zadanie trudne. O ile twórcę agletu można stosunkowo łatwo określić, jeżeli będziemy stosowali mechanizm cyfrowego podpisu kodu apletu, o tyle określenie właściciela jest znacznie trudniejsze. Ze względu na ciągłą zmienność stanu agletu, cyfrowe podpisywanie bloku informacji przechowującego ten stan praktycznie nie wchodzi w grę. W tej sytuacji korzysta się z podejścia organizacyjnego, polegającego na tym, że wszyscy agenci wykreowani i działający w obrębie danej domeny są obdarzani zaufaniem o poziomie takim, jaki ma użytkownik/właściciel agenta. Jeżeli przemieszczając się agent zmienia domeny, jego uprawnienia mogą ulegać zmianie w zależności od tego, z jakiej domeny pochodzi, jakie domeny odwiedził podczas swojej „wędrówki" itd. Oczywiście mechanizm taki ma wiele niedoskonałości, jednak jak to określają znawcy tematu, zapewnia „rozsądny" poziom bezpieczeństwa. Dostępne obecnie systemy mobilnych agentów wykorzystują następujące mechanizmy:

- identyfikacja i weryfikacja autentyczności użytkowników i domen,
- kontrola integralności komunikacji między serwerami danej domeny,
- „drobnoziarniste" mechanizmy autoryzacji oparte na modelu bezpieczeństwa zastosowanego w JDK 1.2.

4.3. Cykl życia agletu

Klasa `com.ibm.aglet.Aglet` definiuje podstawowe funkcje mobilnych obiektów, zaś każdy aglet jest wystąpieniem pewnej podklasy tej klasy. Istnieją dwa sposoby wykreowania nowego agletu. Pierwszą z nich jest utworzenie zupełnie nowego agletu na podstawie danej definicji klasy poprzez wywołanie metody `AgletContext.createAglet(URL codebase, String name, Object init)`. Metoda ta kreuje nowy aplet w obrębie konkretnego kontekstu i inicjalizuje go, jeżeli jest to potrzebne. Następnie woła metodę `Aglet.onCreation(Object init)` nowo utworzonego agletu, przekazując mu jako parametr referencję do obiektu, która została przekazana jako parametr wywołania metody `createAglet`. Drugą metodą stworzenia agletu jest stworzenie kopii istniejącego agletu przez wywołanie metody `Aglet.clone()`. Sklonowany aglet znajduje się w tym samym stanie co aglet oryginalny, jednak różni się atrybutem identyfikującym go – obiektem `AgletID`, a więc posiada odmienną tożsamość. Tuż po sklonowaniu nowy aglet może wykonać dowolną czynność, np. przenieść się na inną maszynę, przejść w stan uśpienia czy też przystąpić do wykonywania określonego zadania. Rys. 4 przedstawia cykl życia agletu.



Rys. 4. Cykl życia agletu
Fig. 4. Aglet life cycle

Aglet może przesyłać się na zdalny serwer przez wywołanie metody `Aglet.dispatch(URL dest)`. Mówiąc bardziej precyzyjnie, aglet działa w obrębie pewnego kontekstu i może się przenieść z jednego kontekstu do innego wstrzymując na czas przesyłki swoje działanie. Ponieważ na jednej maszynie (rozumianej jako komputer) może jednocześnie działać kilka aplikacji pełniących rolę serwera, zaś na każdym serwerze może jednocześnie funkcjonować kilka kontekstów w obrębie jednej maszyny wirtualnej Javy, każdy kontekst jest jednoznacznie opisywany przez następujące parametry składające się na jego adres: adres maszyny (typowo jest to adres IP), numer portu, który obsługuje dany serwer oraz nazwa kontekstu w obrębie tego serwera. Przykładem pełnego adresu kontekstu może być: atp://shadow.eti.pg.gda.pl:1234/my_context

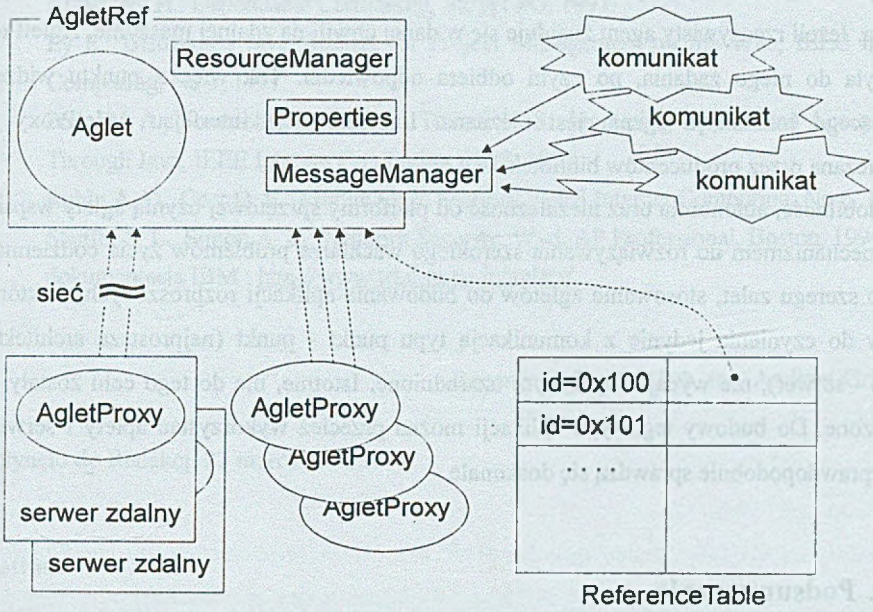
Przesyłanie agletu wiąże się ze wstrzymaniem jego wykonania, serializacją jego kodu i bieżącego stanu to standardowej postaci umożliwiającej ich przesłanie, po czym następuje przesłanie agletu w tej postaci do miejsca przeznaczenia. Po zakończeniu przesyłki po stronie nadawcy likwidowany jest wątek (lub kilka wątków) związany z właśnie wyekspediowanym agletem i następuje zwolnienie wszystkich związanych z nim zasobów. Po stronie odbiorcy z otrzymanego strumienia danych następuje rekonstrukcja agletu, po czym jest mu przydzielany wątek (lub kilka, o ile jest to potrzebne). Oczywiście, aglet może być trwały, tzn. niekoniecznie musi być mobilny.

W odróżnieniu od typowych obiektów Javy, które są automatycznie usuwane przez tzw. garbage collector, aglet, dopóki jest aktywny, może decydować o swoim losie. Jeżeli w celu zabicia agletu wywołamy metodę dispose(), zostanie wywołana metoda onDisposing(), której zadaniem jest zwolnienie zajętych w danej chwili zasobów, takich jak deskryptory plików czy połączenia z bazami danych. Należy zwrócić uwagę na różnicę między opisywaną metodą a standardową metodą finalizer(), która jest wołana przez maszynę wirtualną JAVA'y, kiedy następuje „sprzątanie po niepotrzebnym już obiekcie". W odniesieniu do agletów, to programista jest odpowiedzialny za zwolnienie zajętych zasobów, bowiem w przeciwnym razie mogą one nie zostać zwolnione automatycznie.

4.4. Architektura obiektów klasy *AgletRef*

Obiekt *AgletRef* jest wewnętrzną reprezentacją agletu [11]. Zawiera on wszystkie niezbędne komponenty. Są nimi *MessageManager* zarządzający wiadomościami nadsyłanymi do agletu, *ResourceManager* zarządzający zasobami wykorzystywanymi przez aglet oraz

wszystkimi innymi zasobami nierozerwalnie związanymi z każdym agletem, którymi są np. obiekty *AgletInfo* czy też informacje wykorzystywane przez system bezpieczeństwa. *AgletRef* implementuje większość własności definiowanych przez klasę `com.ibm.aglet.Alget`.



Rys. 5. Struktura agletu
Fig. 5. The structure of the aglet

Widoczna na rys. 5 tablica referencji (ReferenceTable) służy do mapowania identyfikatorów agletów AgletID na ich rzeczywiste lokalizacje. W momencie gdy w danym kontekście pojawia się nowy aglet, tzn. gdy jest on tworzony lub „przybywa" z innego kontekstu, jest automatycznie tworzony nowy obiekt AgletRef, zaś do tablicy referencji dodawana jest nowa pozycja. Wraz z zakończeniem działania agletu lub przeniesieniem do innego kontekstu następuje usunięcie obiektu AgletRef oraz usunięcie odpowiadającej mu pozycji w tablicy referencji.

Z kolei obiekty AgletProxy pełnią rolę uchwytów/wskaźników agletu, przez co umożliwiają dostęp do niego. Klasa Aglet posiada szereg publicznych metod, które ze względów bezpieczeństwa nie powinny być osiągalne bezpośrednio, np. dla innego agletu. W tej sytuacji wprowadzono właśnie klasę AgletProxy, która służy jako interfejs dostępu do agletu. Tak więc każdy aglet, który chce wołać metody innego agletu, musi czynić to poprzez

AgletProxy, pełniącego rolę osłony chroniącej agenta przed atakami z zewnątrz. Co więcej, po wywołaniu przez obcego agenta AgletProxy kontaktuje się z zarządcą bezpieczeństwa w celu sprawdzenia, czy wołający jest uprawniony do wykonania danej metody. Kolejnym ważnym zadaniem interfejsu AgletProxy jest zapewnienie przezroczystości lokalizacji agenta. Jeżeli rzeczywisty agent znajduje się w danej chwili na zdalnej maszynie, AgletProxy przesyła do niego żądania, po czym odbiera odpowiedzi. Tak więc z punktu widzenia wołającego lokalizacja agenta jest nieznana. Implementacja interfejsu AgletProxy jest dostarczana przez producentów bibliotek JAAPI.

Mobilność, autonomia oraz niezależność od platformy sprzętowej czynią aglety wspaniałym mechanizmem do rozwiązywania szerokiego wachlarza problemów życia codziennego. Mimo szeregu zalet, stosowanie agletów do budowania aplikacji rozproszonych, w których mamy do czynienia jedynie z komunikacją typu punkt - punkt (najprostsza architektura klient - serwer), nie wydaje się niczym uzasadnione. Istotnie, nie do tego celu zostały one stworzone. Do budowy tego typu aplikacji można przecież wykorzystać aplety i serwlety, które prawdopodobnie sprawdzą się doskonale.

5. Podsumowanie

Bezpieczeństwo systemu rozproszonego można uzyskać poprzez wprowadzenie różnego typu mechanizmów ochrony i zapewnienia autentyczności i integralności. W pracy ograniczono się jedynie do poziomu aplikacji, analizując podstawowe elementy konstrukcyjne programów użytkowych implementowanych w nowoczesnym, intensywnie rozwijanym języku Java. Tego typu konstrukcje zasadniczo wpływają zarówno na metodologie wytwarzania oprogramowania, jak i na politykę bezpieczeństwa systemów.

LITERATURA

1. Adida B.: Security the Web, IEEE Internet Computing, No 4, 1997.
2. Anupam V., Mayer A.: Secure Web Scripting, IEEE Internet Computing, No 6, 1998.
3. Evans E., Rogers D.: Using Java Applets and CORBA for Multi-User Distributed Applications, IEEE Internet Computing, No 3, 1997.

4. Hashii B., Pandery L. R., Samordin S.: Securing Systems Against External Programs, IEEE Internet Computing, No 6, 1998.
5. Karjoth G., Lange D. B., Oshima M.: A Security Model for Aglets, IEEE Internet Computing, No 4, 1996.
6. Krawczyk H.: Dependable Computing, Skrypt PG, 1997.
7. Ly E.: Distributed Java Applets for Project Management on the Web, IEEE Internet Computing, No 3, 1997.
8. Puliafito A., Tomarchio O., Vita L., Trivedi K. S.: Increasing Application Accessibility Through Java, IEEE Internet Computing, No 4, 1998.
9. Rubin A. D., Geer D. E.: Mobile Code Security, IEEE Internet Computing, No 6, 1998.
10. Shaffer S. L., Simon A. R.: Network Security, Wyd. AP Professional, Boston, 1994.
11. dokumentacja IBM : http://www.trl.ibm.co.jp/aglets/

Recenzent: Prof. dr hab. inż. Andrzej Grzywak

Wpłynęło do Redakcji 23 marca 1999 r.

Abstract

In the paper we consider multi-user distributed applications (see Fig.1) that were built using Java applets, servlets and aglets. Java is a relatively new object-oriented programming language gaining rapid acceptance among software developers. Java source code can be compiled into a machine independent format consisting of virtual machine instructions and symbolic data (a byte-code format). Upon execution this code must be interpreted by a special Java interpreter, a part of Java runtime. This interpreter consists of parts required host and operating system platform. One of the ways of implementing user-site client software is using Java applets. Users transparently download the applets when they are needed thus removing the need to manually distribute and install any aplication specific software (see Fig.2).

Security requirements are twofold:

1. Information should be available only to those meant to see it.
2. Actions should be performed only by those authorized to perform them.

Very important problem is an applet authentication. The Java language and runtime have built-in security hooks. The Java runtime built-in authentication mechanism can determine

whether an applet is allowed to perform a particular action. Besides, digital signature based on public key cryptography can be used. Using a private key, a supplier of applet software can sign the application's executable byte-code. When the user downloads the applet, the Java Security Manager in the browser's runtime uses the supplier's public key to verify that the signed applet really comes from the trusted entity, and then allows the applet to perform an action.

Another solutions use servlets. Their functionality is similar to CGI programs (see Fig. 3). Besides a simple user interface they have a lot of advantages including efficiency, simplicity and good security as well as some other important features. As it was shown, servlets can be used as the alternative for applets for some type of Web applications. Nevertheless, in practice we can use both of them taking into account their advantages and disadvanteges. In this case it is easier to achieve a good performance of the application, reduce net traffic and build complex, fully functional and user-friendly user interface. Moreover, security restrictions used for servlets are considerably reduced. The most important facility making us to use servlets instead of applets is that, in general, servlets allow to access remote resources and access local files.

Mobile agents offer a new paradigm for distributed computation, but their potential benefits must be weighted against real security threats they pose. Java-based environment also allows to build mobile agent applications. Then special constructions called aglets can be used. Aglets are serialized Java objects that visit aglet-enabled hosts in a computer network. An aglet that executes on one host can halt execution, dispatch to a remote host and resume execution there. When the aglet migrates, it takes along its program code as well as its data. Typical aglet life cycle is shown in Fig. 4. Fig. 5 shows aglet structure with essential, external elements from aglet's environment. An aglet might be unaware of the security policy of the hosting context and how it is enforced. The user can be authenticated prior to creating the aglet, security is then enforced automatically. Some aglets will need to control or influence policy stategy enforced by the system on their behalf, but will not enforce it themselves. Others will need to enforce their own security methods to control access to their own data or to audit their own security-relevant activities. The paper describes both approaches.

In summary, it was shown that using applets, servlets or aglets we can individually create both, the application development strategy and security services in distributed information-processing systems.