Security-enabled code deployment for heterogeneous networks

Stamatis Karnouskos

Fraunhofer Institute for Open Communication Systems
Kaiserin-Augusta-Allee 31, D-10589 Berlin, Germany
Stamatis.Karnouskos@fokus.fraunhofer.de

Abstract

Future services on converged heterogeneous networks are expected to increase the demand for code distribution, on the fly installment and dynamic management. Component discovery, download, installation, revocation, and profiling are critical in supporting next generation enabling technologies and application evolvement. We present here the architecture of an active component manager and discuss on metadata that should be used to describe components and their capabilities. Furthermore we elaborate on XML digital signatures as an integral part of secure component distribution. Finally we investigate topologies for component distribution and how emerging technologies like peer-to-peer could be used in the context of active component manager. This paper is motivated by the vision of Google-like code discovery, and flexible management of it, within the future semantic web context.

Keywords: *XML Digital signatures, component deployment, active networks, Dublin Core, Resource Description Framework, metadata, code discovery.*

1 Introduction

Reduction of the time-to-market services, as well as their customization lead to the introduction of the programmability in the network elements. Today we witness integration between the services offered in telecom networks and those in data networks, driven mainly by a service-oriented market that seeks granularity and interoperability. The need will become more evident in 3G and beyond networks (as envisioned also in the WWRF book of vision - http://www.wireless-world-research.org/general_info/BoV2001-final.pdf) where users and their session might roam over wired, wireless and mobile networks.

Active and programmable networks [4] introduce a new network paradigm where network-aware applications and services can be not only distributed, but also can configure the heterogeneous network to optimally respond to a task's requirements. We are able to utilize within the network a) computation as we are able to compute on data received from active nodes and b) programmability, as we can inject user code into the network nodes in order to realize customized computation. Being able to achieve the above, we succeed in decoupling network services from the underlying hardware, deploy fine-grained customized services, relax the dependencies on network vendors and standardization bodies and generally open the way for higher level network-based application programming interfaces.

However, as active networks promote user injected code deployment, they also reveal an infrastructure that is far more vulnerable than the current passive networks if not protected appropriately. Security and trust management in such a heterogeneous environment becomes an extremely sensitive issue. In this paper we will analyze the active code deployment, and present an approach on its secure handling based on digital signatures, public key infrastructure and state of the art cryptographic protocols and algorithms.

2 Motivation

Current active network research is heading towards component based execution environments [5] that can be dynamically configured. In such an infrastructure the need to discover, update, reuse existing code and bind components[1] is fundamental. Therefore, future services and networks are expected to increase the demand for code distribution and management. This is not limited only to Internet services and related technologies such as active/programmable networks but extends to the general area of mobile user support context, especially in 3G and beyond infrastructures. As depicted in Figure 1, the user has a "user context" to which several devices are attached with a high heterogeneity in hardware and software. All these devices are expected to host more or less an execution environment (EE) of some kind, where code executes providing the ability to build

[1] Components are seen as a collection of generic code entities with dependencies. Components and code are interchangeable meanings within the context of this paper.

sophisticated services. The code to be used as such, resides in Internet code repositories (e.g. open source repository SourceForge - sourceforge.net), in legacy web and ftp servers, in Usenet archives etc, depending on the mean that the author has chosen to distribute his code. As now the user can download, inject and execute code (on its PDA, Java enabled mobile phone or in an active node) it is expected that the following matters will arise with an exponential growth:

- The need to find code based on criteria e.g. author, execution environment, platform, technology, description, performance, etc
- The need to describe the developed code based on widely acceptable templates and vocabulary. Using these, the vision of semantic web is promoted and also the automatisation of tasks such as search, management etc can be delegated to intelligent technologies e.g. intelligent mobile agents.
- The need to provide a way of making widely known the code existence. Nowadays there is wide variety of code available on the Internet, which remains mainly unknown due to the fact that there is no standardized description of it, nor code-specific search engines that can parse its metadata based description.
- The need to provide indexing of code modules available e.g. on the Internet based on best effort (for code that does not provide metadata about its description – something done by all search engines today for legacy web pages) and on specific semantics that the authors, managers etc of the code provide.
- The need to integrate security and trust from day one within all the above matters, and possibly in a later step introduce digital rights management (DRM).

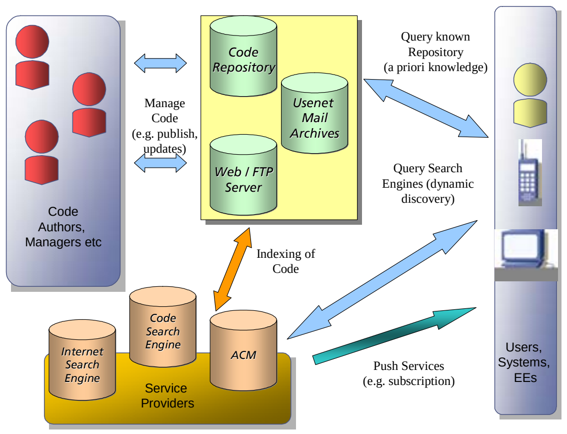

Code Authors, Managers etc
Manage Code (e.g. publish, updates)
Code Repository
Usenet Mail Archives
Web / FTP Server
Query known Repository (a priori knowledge)
Query Search Engines (dynamic discovery)
Indexing of Code
Internet Search Engine
Code Search Engine
ACM
Service Providers
Push Services (e.g. subscription)
Users, Systems, EEs

Figure 1 - Code Distribution

Special care has to be taken with regard to the security concerns that arise with component/code distribution. Active networks promote distributed services and ease the network code deployment. This code needs to be found, fetched, installed and managed. As downloading and on the fly installing code on running systems will be a future commodity, we have to make sure that malicious parties do not take advantage of these capabilities. Furthermore a single point of code distribution such as the web site of the distributor, might not be a viable case when millions of nodes try to download the updates. Therefore trust needs to be inserted in the code package, which can be then distributed by third parties and simultaneously allow the user to verify that this is the same as the original release. Therefore we have to harden with security mechanisms all steps involved in an active code deployment scenario. A detailed analysis on the threat model as well as the security requirements of the active networks can be found in [1], where also various solutions are proposed. However here we do not focus on mobile code (e.g. agents), but we rather take a general approach on the matter of code distribution in open programmable infrastructures such as active networks. For the security part we have to take care of the following fundamental principles:

- **Integrity:** Data consistency should be maintained. Compromised components jeopardize the stability, safety and security of the systems.
- **Authentication**: We must have the capability of getting the credentials of the entities related to the code and be able to validate these credentials against some authority. Successful validation speaks for an authenticated entity and is further used for authorization decisions.
- **Authorization:** We must be able to make policy-based decisions on the deployment of code based on the authenticated entity. Only authorized entities should be able to deploy and manage the components.
- **Non-repudiation:** The author of the component should not be able to deny authoring the specific component or even its expected behaviour.
- **Confidentiality:** If it is desired, no one else can access or copy the data related to a component.

Furthermore we need to investigate technologies, topologies and approaches so that the code deployment is done in the most flexible but also promising way in order to be open for future application needs. Since active networks offer such a programmable infrastructure, our approach is based on their characteristics for proof of concept, but clearly the target is broader and tackles generally selective code deployment issues in heterogeneous networks.

3 Active Component Manager

As users are allowed to inject components within an active node, we realize that code deployment is a fundamental issue within the active network community. Therefore, somehow all AN approaches are expected to feature an Active Component Manager (ACM) or its logic at node or even EE level. The basic functions (also depicted in Figure 2) include:

- **Installation :** The Uniform Resource Identifier (URI as defined in RFC 2396) of the component to be installed is passed to the ACM. The ACM fetches and installs the requested component if it doesn't already exist in its local database (DB).
- **Deinstallation:** The requested component is removed from the ACM DB.
- **Retrieve:** The requested component that is stored in the ACM DB or its profile is returned to the requestor. This could be done for several reasons e.g. if one wants to transport it to another node that does not feature an ACM compatible API. In that case a mobile agent implements the missing functionality and transports the component from node to node. Another possibility would be to have a stationary agent that wraps the missing functionality, is permanently active on the remote node, and handles the ACM communication. These are design decisions based on user's requirements.
- **Search:** The ACM DB is searched for existing versions of components. The search is done based on fields defined in the component's profile as presented in section 4. When an *Index* is requested the ACM returns all visible for the requesting entity components (policy-based authorization) existing locally. Depending on the search parameters, the ACM can use this interface to search other ACMs on neighbouring nodes in order to find nearby code (function similar to peer-to-peer networks).
- **Notification:** This is used for a twofold purpose. Firstly, the ACM is able to accept external notifications for the components it hosts e.g. when a new version is available. Secondly, it is possible to allow local users or external ACMs to subscribe and get notified for events like code removal, code upgrade etc. This allows mirroring of ACMs which is useful for networks with identically configured active nodes.
- **Update:** This interface is used for automatic upgrades. The request issued here triggers the ACM to search if a new version of the specified component is available on the Internet. Assuming that the new component version is backwards compatible, the ACM can remove existing versions, install the new one and notify the affected parties.

The ACM has an interface available for communicating with requests coming from the execution environments. All requests are authorized against the policies that exist for the local node via a security manager. The security manager evaluates both the policies of the local node and the policies of the execution environment that the request is coming from. Further entities that are bound to the security manager include the credential manager whose task is to verify the security credentials supplied either via a request at ACM (e.g. the user *k* requested to install component *x* in execution environment *n*) or from ACM (e.g. verify that the signer of component *x* has a valid certificate). The Audit manager handles all logging requests while the resource manager takes care of the resource management requests (e.g. if the profile of the component sets the minimum disk space to be used at 10 MB and the available disk space is only 8 MB – the component installation fails due to resource mangle). The resource manager is expected to do even more interesting things like real-time monitoring of the consumed resources and enforcement of resource usage policy. Some early thoughts on the generic relations and functionality of this architecture can be found on earlier work [2]. Finally all components and their profiles are stored in one or more databases e.g. the component is stored in a relational DB but its RDF description in an RDF-capable DB (http://www.w3.org/2001/05/rdf-ds/DataStore) accessible via the ACM.

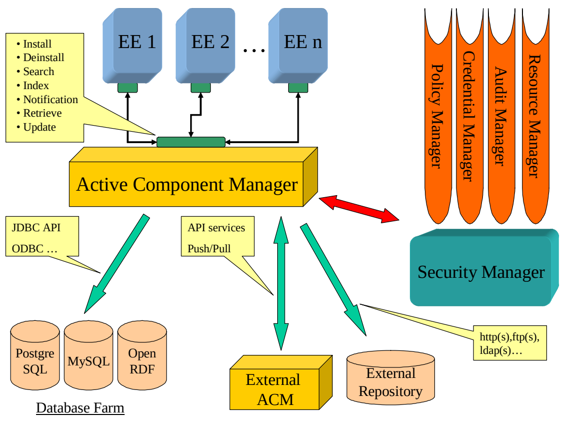


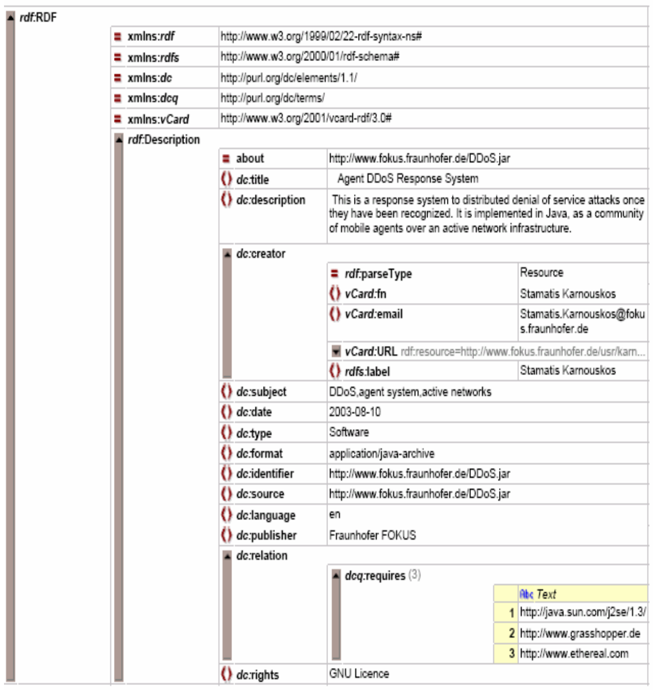
Figure 2 – Active Component Manager Architecture

In order not to bring the node in an unstable state e.g. when a component is removed while there still exist services that use it, we have two directions. The simplest one installs a component for each user that requests so. Although this is flexible, it doesn't scale well as in an active node with *x* users we have *x* times the same component. Therefore the ACM installs only once a specific component and keeps a list of users that use it. Even if one user deinstalls a specific version, this is marked as inactive for the user but is not actually removed from ACM. The actual removal takes place when the last user of the list requests also that this component should be deinstalled. Different versions of the same component are handled as different components when it comes to installation/deinstallation.

As the ACM maintains a list of the components installed locally and is able to query other ACMs, we are able to build a living network of ACMs that are able to exchange component profiles as well as the component code. In this respect of dynamic discovering and downloading components the peer-to-peer technology seems appealing as explained in section 7. Finally via the Update interface we are able to realize automatic upgrades of components. This is vital in keeping our node up-to-date with the latest releases and bug-fixes of software. Since the existing approach is primitive (deinstalls the old versions, installs the new component, makes it the default one and notifies the affected parties) we are not able to realize sophisticated upgrades where dependencies exist or even better online upgrades (e.g. the OMG approach as depicted in http://www.omg.org/cgi-bin/doc?orbos/02-01-01), but existing solutions could be integrated. Despite that, it seems that ACM can be one of the core components upon which more sophisticated approaches can be built.

4 Metadata for Component Distribution

It is not cool to be different, at least not when code distribution in Internet is concerned. The vision of future Internet incorporates the seamless component distribution, indexing and integration. Unfortunately, this is not the case today. Once a component is developed and needs to be publicly available, one comes up to the obstacle of describing his component's capabilities in a universally acceptable way. Therefore we need metadata for the component in an expressive and extensible way that will also cover future needs.



```
rdf:RDF
    xmlns:rdf      http://www.w3.org/1999/02/22-rdf-syntax-ns#
    xmlns:rdfs     http://www.w3.org/2000/01/rdf-schema#
    xmlns:dc       http://purl.org/dc/elements/1.1/
    xmlns:dcq      http://purl.org/dc/terms/
    xmlns:vCard    http://www.w3.org/2001/vcard-rdf/3.0#
    rdf:Description
        about            http://www.fokus.fraunhofer.de/DDoS.jar
        dc:title         Agent DDoS Response System
        dc:description   This is a response system to distributed denial of service attacks once
                         they have been recognized. It is implemented in Java, as a community
                         of mobile agents over an active network infrastructure.
        dc:creator
            rdf:parseType    Resource
            vCard:fn         Stamatis Karnouskos
            vCard:email      Stamatis.Karnouskos@foku
                             s.fraunhofer.de
            vCard:URL rdf:resource=http://www.fokus.fraunhofer.de/usr/karn...
            rdfs:label       Stamatis Karnouskos
        dc:subject       DDoS,agent system,active networks
        dc:date          2003-08-10
        dc:type          Software
        dc:format        application/java-archive
        dc:identifier    http://www.fokus.fraunhofer.de/DDoS.jar
        dc:source        http://www.fokus.fraunhofer.de/DDoS.jar
        dc:language      en
        dc:publisher     Fraunhofer FOKUS
        dc:relation
            dcq:requires (3)
                    Text
                 1  http://java.sun.com/j2se/1.3/
                 2  http://www.grasshopper.de
                 3  http://www.ethereal.com
        dc:rights        GNU Licence
```

Figure 3 - Description of DDoS Component

Today there are some efforts guiding towards the universal component description, but are mostly stand-alone, site-specific, too complex or not widely used templates. E.g. iBiblio.org (former sunsite.unc.edu) requires from all Linux software package developers to fill-in the Linux software map entry template (http://www.ibiblio.org/pub/linux/LSM-TEMPLATE). Others like freshmeat.net (www.freshmeat.net), Trove (http://www.catb.org/~esr/trove/), provide similar text-based descriptions or even stand-alone XML such as Portable Application Description (PAD - http://www.asp-shareware.org/pad). These efforts provide some basis but the descriptions are ambiguous and obsolete. The Open Software Description Format (OSD - http://www.w3.org/TR/NOTE-OSD) is XML-based and features "a vocabulary used for describing software packages and their dependencies for heterogeneous clients" and comes a step closer to the target. The Open Source Metadata Framework (OMF - http://www.ibiblio.org/osrt/omf) aims at describing data (metadata is data that describes data) about Open Source documentation so that they can be easily used by applications like ScrollKeeper (scrollkeeper.sourceforge.net). OMF uses XML and the Dublin Core Metadata Initiative (DCMI - www.dublincore.org). The Dublin Core metadata standard is widely used and consists of a simple element set for describing a wide range of networked resources. Furthermore the Dublin Core vocabulary can be used to define additional semantics about the resources described within a Resource Description Framework (RDF - http://www.w3.org/RDF) fragment. RDF co-evoluted with DCMI and can be seen as complementary approaches within the web's metadata architecture.

RDF or a similar more expressive schema language like DARPA Agent Markup Language (DAML - www.daml.org) is expected to form the foundation of the Semantic Web (http://www.w3.org/2001/sw) vision. XML forms the basis by being the transport schema, while RDF, DAML and alike, provide the information representation framework. XML is an expressive language that has a well defined grammar for defining message structures accompanied with many tools for generating, consuming and working with XML documents. OSD and OMF set a good basis for partially providing what we need. However since as standalone solutions none of them is sufficient, we need to combine their abilities and extend them in order to cover emerging requirements. Although the RDF/DCMI combination (including the DCMI's element refinements) are probably enough to describe the component distribution context, no general template up today exists in order to be used by the community.

Figure 3 describes in XML/RDF a software prototype that we developed. The Dublin Core vocabulary is used to define additional semantics about the resources described within the RDF fragment. The prototype is a Distributed Denial of Service (DDoS) Attack response system and as it can be seen useful info are contained within the XML description such as name, description, author's contact details, download location, other software packages that it might require such as Java, Grasshopper mobile agent platform (www.grasshopper.de) and ethereal network analyzer (www.ethereal.com). This profile is by no means complete but is used here as a proof of concept. In the future a more complete profile specifically for describing the software packages and active node specific characteristics has to be developed and used. Now if a metadata-capable search engine crawls the site and parses this profile, it will be able to accept requests like "search for all components authored by Mr. Karnouskos", or more complex ones like "search for all java-based v1.3 and above software that requires ethereal". An intelligent agent that parses this profile could also check the "requires" section and not only fetch the DDoS component but also fetch the required packages that are not installed locally. Once one has the capability to describe in a machine-readable way richly enough his components, there will be many others that will invest the required time to make intelligent decisions based on the data they are able to acquire. This brings us one more step closer into making a reality the vision of the Semantic Web, and in our case for active and programmable networks at node or even EE and active application level.

5 XML Digital Signatures

Digital signatures can be used for

- Identification
- Proof of involvement in the act of signing
- Associate the signer with a document
- Provide proof of the signer's involvement with the content of the signed document.
- Provide endorsement of authorship.
- Provide endorsement of the contents of a document authored by someone else

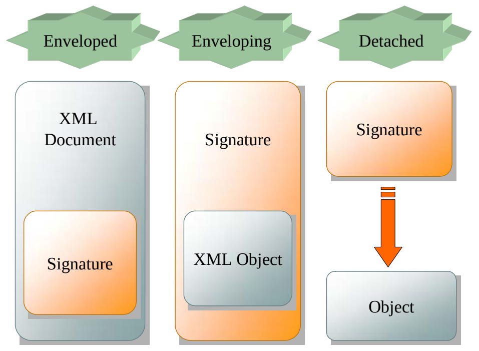


Figure 4 – Arts of XML Digital Signatures

The XML signature is a method of associating a key with referenced data. XML Signatures can be applied to any digital content (data object), including XML. An XML Signature may be applied to the content of one or more resources. Signed data can be located within the XML that includes the signature or elsewhere. Based on this location, there are three different types of signatures (depicted in Figure 4) i.e. Detached, Enveloping and Enveloped signatures:

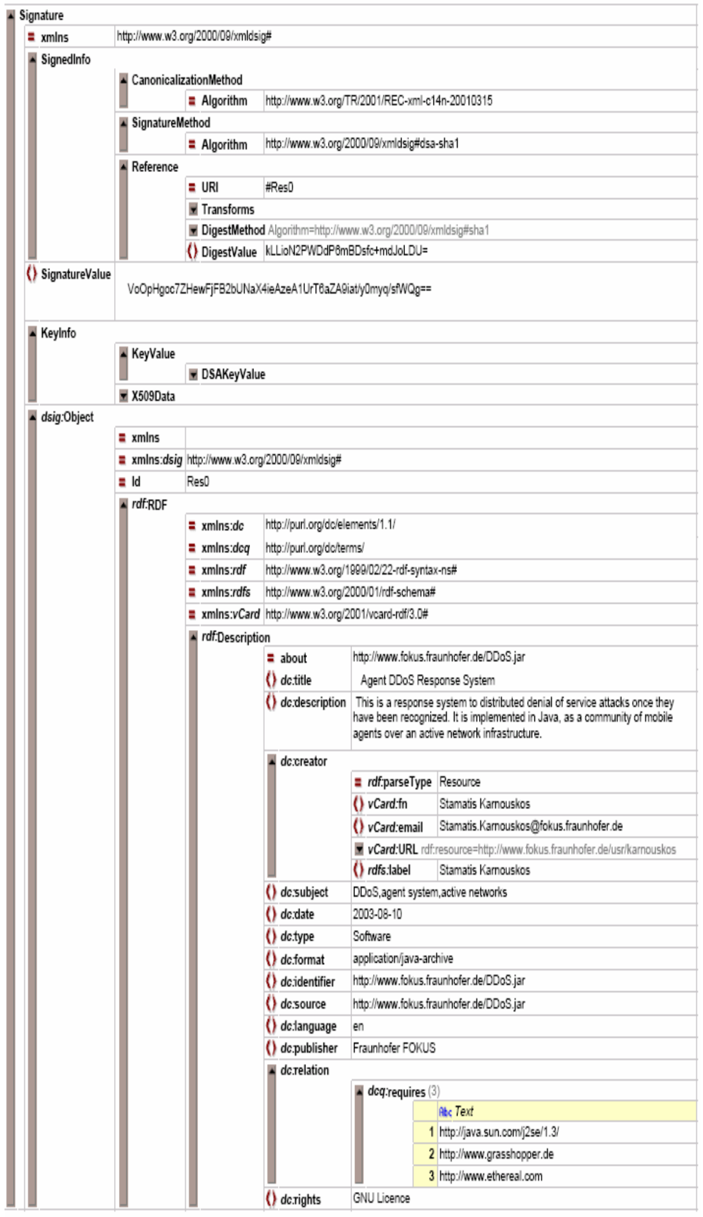
- An enveloped signature is enclosed inside the XML element it signs. The enveloped signature must take care not to include it's own value in the calculation of SignatureValue;
- Enveloping signatures are over data within the same XML document as the signature; An enveloping signature is a signature, which includes the object to be signed within it, and identifies it via an URI or a transform (i.e. XPath);
- Detached signature is a signature over a content external to the signature element and is identified through an URI. This description applies also if the signature and the object reside in the same XML document as sibling elements.

Enveloping and detached signatures seem more fit for our purpose. Generally in component distribution, digital signatures can be deployed with

a) the component profiles that float around the network or
b) the component package itself.

In the semantic web it is expected that descriptions of components similar to the one presented in Figure 3, are freely distributable among interested parties, copied in non author controlled sites and are indexed by search engines such as HotMeta (http://www.dstc.edu.au/Research/Projects/hotmeta) that are able to parse the metadata they contain. We have to make sure that these component profiles are not altered in any non-intended way, be able to traceback the author of the profile and make authentication/authorization decisions based on the data that we extract from it. Therefore one possible way is to sign the XML file by using digital signatures.

Towards this direction work is done within the WWW consortium. Specifically, XML digital signatures (http://www.w3.org/TR/xmldsig-core) can be deployed to guarantee integrity over untrusted networks such as the Internet, help with the authentication, authorization and the non-repudiation activities. Figure 5 depicts the digitally signed version of the XML depicted in Figure 3. We have used IBM’s Security Suite (http://alphaworks.ibm.com/tech/xmlsecuritysuite) and X509 Certificates. The new XML file (depicted in Figure 5) integrates all info from the original one and contains also a X509 certificate of the user that signed it. One can extract the certificate, verify it and use its data for authorization decisions. This is even more powerful in case of attribute certificates that bring in the benefits of a privilege management infrastructure [6]. In the approach demonstrated above we have signed the component profile and protected it. But what happens if one of the resources changes e.g. the binary gets corrupted? It is also possible to actually sign the exact location of the component and its other network resources by calculating the digest of each resource, including them in the final XML file and signing it. In this way we make sure not only that the component profile is safe but also the links that it contains to resources (e.g. binary code) have also not changed. This of course has the obvious side-effect, that the profile is tied up to specific versions of external resources and if e.g. a new version of a component is released (which hopefully is backwards compatible) the profile will still point to the older version of it. Practically that means that the verification for this specific changed resource will fail, as it is different from the original one that was used when the component profile was signed. Therefore a different URI needs to be used for each component. The latest version could be retrieved by other means e.g. a static location of a profile and the respective URI that point always to the latest available version. Alternatively the component could be also integrated in the same XML file having its profile, but this might create bandwidth problems since most requests are expected to target the informational data of the profile



```
Signature
  xmlns               http://www.w3.org/2000/09/xmldsig#
  SignedInfo
      CanonicalizationMethod
          Algorithm   http://www.w3.org/TR/2001/REC-xml-c14n-20010315
      SignatureMethod
          Algorithm   http://www.w3.org/2000/09/xmldsig#dsa-sha1
      Reference
          URI         #Res0
          Transforms
          DigestMethod Algorithm=http://www.w3.org/2000/09/xmldsig#sha1
          DigestValue kLLioN2PWDdP8mBDsfc+mdJoLDU=
  SignatureValue
      VoOpHgoc7ZHewFjFB2bUNaX4ieAzeA1UrT6aZA9iat/y0myq/sfWQg==
  KeyInfo
      KeyValue
          DSAKeyValue
      X509Data
  dsig:Object
      xmlns
      xmlns:dsig      http://www.w3.org/2000/09/xmldsig#
      Id              Res0
      rdf:RDF
          xmlns:dc    http://purl.org/dc/elements/1.1/
          xmlns:dcq   http://purl.org/dc/terms/
          xmlns:rdf   http://www.w3.org/1999/02/22-rdf-syntax-ns#
          xmlns:rdfs  http://www.w3.org/2000/01/rdf-schema#
          xmlns:vCard http://www.w3.org/2001/vcard-rdf/3.0#
          rdf:Description
              about           http://www.fokus.fraunhofer.de/DDoS.jar
              dc:title        Agent DDoS Response System
              dc:description  This is a response system to distributed denial of service attacks once they
                              have been recognized. It is implemented in Java, as a community of mobile
                              agents over an active network infrastructure.
              dc:creator
                  rdf:parseType  Resource
                  vCard:fn       Stamatis Karnouskos
                  vCard:email    Stamatis.Karnouskos@fokus.fraunhofer.de
                  vCard:URL rdf:resource=http://www.fokus.fraunhofer.de/usr/karnouskos
                  rdfs:label     Stamatis Karnouskos
              dc:subject      DDoS,agent system,active networks
              dc:date         2003-08-10
              dc:type         Software
              dc:format       application/java-archive
              dc:identifier   http://www.fokus.fraunhofer.de/DDoS.jar
              dc:source       http://www.fokus.fraunhofer.de/DDoS.jar
              dc:language     en
              dc:publisher    Fraunhofer FOKUS
              dc:relation
                  dcq:requires (3)
                           Text
                         1 http://java.sun.com/j2se/1.3/
                         2 http://www.grasshopper.de
                         3 http://www.ethereal.com
              dc:rights       GNU Licence
```

Figure 5 - Digitally Signed Component Profile

and few of them will go one step further in downloading the code, therefore is seems a better idea to keep component profiles and their actual implementations apart.

Furthermore the XML encryption (http://www.w3.org/TR/xmlenc-core) can be used in order to keep secret the component description (in whole or partially) and make them available only to selected parties. XML encryption can even address areas that are not covered by SSL/TLS, Internet's de facto communication standard, namely a) encrypting part of the data being exchanged and b) secure sessions between more than two parties. By combining XML Encryption with XML Signature we can provide both message digest and message authentication functionality.

IBM's Security Suite implements also the XML Access Control (XAC) which aims at providing XML documents with a sophisticated access control model and access control specification language (XACL). This makes it possible via the access control policies to control how an XML document appears. The policies also ensure that the XML document is securely updated as specified by the security programmer. In our case this means that via authorization architecture we firstly are able to specify who is actually able to see which fields within our XML document, namely the component profile. Being able to control per XML element authorization requests, we introduce both security and flexibility within our approach.

6 Topologies for ACM

The last years Internet has exploded into the largest decentralized computer system. Various architectures have emerged that are based on the two edge approaches namely the centralized and decentralized ones. ACMs are considered to be the component distribution marketplaces within the AN community and therefore their topology matters. Learning from the Internet distributed computing paradigm we can distinguish the following:

Centralized: Similar to existing approaches, the code is fetched from a central location. This location is usually controlled by the author of the code or by a trusted entity within a network domain (e.g. network administrator). The code is placed there so that clients can request it. Therefore the location of the code as well as the protocol that it can be used for its retrieval is usually a priori known. This approach offers easy management and can be easily secured. The big drawback however it that it is not fault tolerant and difficult to extent.

Decentralized: More than one servers host the code to be fetched. The client may have a reference to the code's location but this is relative and the actual location of the code is resolved the moment the client makes the request. This could be done for several reasons, including load balancing, survivability, network capability etc. With this approach we can take a step further and even keep the protocol that can be used for the downloading of code relative (selection from different URIs). This approach offers fault-tolerance and is easily extensible. Unfortunately it is difficult to manage and tends to be insecure in the sense that it is easy for a node to join the network and provide malicious data. Therefore an extra security level has to be issued here in order to have basic services e.g. authentication. This doesn't necessarily have to be the case in centralized systems where one can verify the node one connects to, and extent this trust to the components downloaded in a secure way from this node.

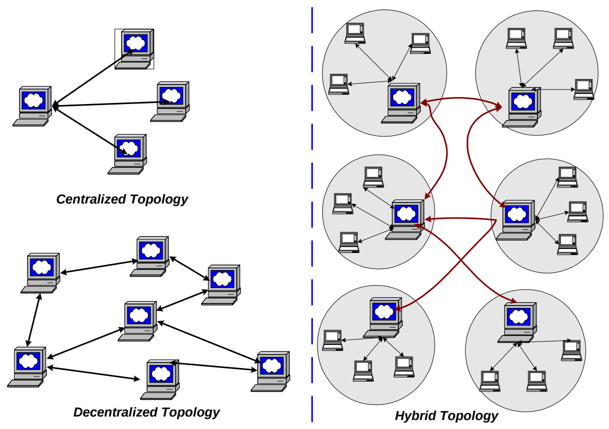


Figure 6 – ACM Network topologies

Hybrid: Here we refer to all the multi-layered approaches that lie between the centralized and decentralized ones. The combinations of existing centralized, decentralized, ring and hierarchical topologies in different layers are infinite. Each one of these could have pros and cons, depending on the environment in which they are deployed. For instance in Figure 6 we have secondary nodes that are connected in a centralized way to one main node. The main nodes however are connected in a completely decentralized way.

The Active Component Manager as presented in this paper supports all above approaches. However the hybrid approach seems more interesting. The ACM stores in its DB info about existing code. The code could be locally stored (and accessed via the filesystem utilities) or rely in an external location. The later allows a plethora of protocols to be used for the code downloading such as ldap(s), http(s), ftp(s) etc. Furthermore instead of having a fixed external location, the ACM could point to another ACM hosted in another AN node within the network. Therefore we can create a virtual web of ACMs (e.g. in P2P style) where entries in their database are hyperlinked. This is a hybrid form could be pushed even further by having the ACMs exchange queries with each other in the effort of trying to find a specific component. This task is eased by the fact that each ACM has an index of all components that it knows, and can be queried via the search interface. By linking the ACMs together we create a web of components. The search function of each ACM can be conceived as a locally based search machine. We can foresee in the future network-wide services that implement a sort of "search engine" for components.

7 Peer-to-peer (P2P) for ACM

Active and programmable networks promise computation in distributed networks. Within this context, P2P is a promising technology that can be applied in the component deployment. Although P2P isn't exactly new[2], as architectures more that two decades old could be today hosted under this label, is gaining momentum. P2P is a set of technologies that enables direct exchange of services and data between computers. The successful examples of distributed file servers like Freenet (freenet.sourceforge.net) and Gnutella (www.gnutelliums.com) could be used within the ACM context. The idea is that ACMs act as P2P nodes and create their own P2P overlay networks. This offers enhanced flexibility for code deployment.

First of all, metadata that describes each file and the elements within it holds the key of success. This should be done in XML as it is a good foundation because it offers a flexible syntax. It gives us the capability of creating customized schemas that structure our content the way we want it. XML schemas can be created by standardization committees, but they can also be made by individuals or communities for widespread usage. By having this flexibility we can bypass lengthy standardization procedures when these schemas are widespread. ANs do exactly the same thing when it comes to deployment of new protocols (by agreeing on abstract computational model one does not have to be protocol specific). RDF/DAML and XML-based protocols for real-time messaging and presence notification like Jabber (www.jabber.org) are particularly promising ways to deploy metadata, but communities must agree on tags. Finally the current state of P2P is evolving into the hybrid approach described above that takes advantages of the pros and cons of centralized and decentralized approaches, i.e. Gnutella now has superpeers, Freenet provides gateways and JXTA (www.jxta.org) search creates a hierarchy of servers for better efficiency. The same will be true for the P2P-based network of ACMs. The ACMs will be able to discover new software releases from the components they possess and upgrade themselves. They also will be able via Instant Messaging (IM) to be passively notified for bugs and actions to be taken. Not rare is the phenomenon where one of the entities involved in the component distribution context either as user, implementer or distributor, wants to revoke component or replace it with a new version that has several bugs fixed. E.g. the administrator that discovers a malicious version of a specific component that is deployed in the network issues an IM notification and all nodes deinstall the specific software. Such capability will give birth to new approaches where the patches or updates are directly pushed to nodes by authorised entities, the moment they are released in a distributed way. Viruses, worms, vulnerability attackers etc today getting more sophisticated and we are heading towards zero-day attacks [7]. Therefore we will need to act in a very short time and update an immense number of nodes, which will result in heavy network load if this is handled by a single site and possibly create bottlenecks within the network. By using P2P technology, better load balancing could be achieved, but then the problem of trust on the updater arises, since we do not want to intentionally allow attackers to insert Trojans in our nodes. The approach presented here partially covers these issues, since the update instruction can come from an authorized entity (e.g. a Cyber Center for Disease Control as proposed in [8]) and the patch can be distributed by third parties and verified by the end users. Therefore a P2P-enabled ACM would allow us to act proactively and quickly when new vulnerabilities are found and apply on-demand patches in a significantly shorter time-frame to a greater network segment even without user intervention and without waiting for the user to query single site (which might be too late). Finally, in a more intelligent scenario users will be able to search metadata capable engines or directly the ACMs that are online and discover new software that they can integrate. For the developer also this is the ideal way to distribute and make widely known the software component he has developed.

8 Discussion

We have been commenting to the design decisions while describing code/component distribution, discovery and management. Special care was taken to bring forward the security concerns, and we have commented on the future directions and possible impact such an approach might have.

[2] Back in 1981, IBM began to introduce communication standards that developed into a peer-oriented network architecture called Advanced Peer-to-Peer Networking (APPN), a significant change from the traditional top-down hierarchical Systems Network Architecture (SNA) model.

8.1 Related work

Existing efforts like Enterprise Java Beans (http://java.sun.com/products/ejb/), .NET (http://www.microsoft.com/net/) and CORBA Component Model (CCM - http://www.omg.org/technology/documents/formal/components.htm) tackle a more extended area; that of service deployment where the components are instantiated and realize services. We address a more narrow area which is part of the above approaches namely only the component distribution one. The component dependency problem is tackled partially, meaning that one component in order to function will need another one installed locally and this can be done dynamically. The network wide dependencies are not investigated and give avenues for future work.

The digitally signed code approach for active networks exists also in other approaches e.g. PANTS [3] but the code itself or its identifier is carried within the capsule. This approach introduces not only interoperable freely distributable secure profiles for describing the components and their capabilities, but also dynamic component discovery, upgrade and leaves to the user the topology of the ACMs in order to better suit his needs.

8.2 Benefits

We believe that the approach presented here provides the foundation for a scalable, robust, extensible solution that tackles current interoperability problems and is open to future challenges. The major benefits provided are:

Middleware for Component Distribution: The ACM as presented here forms a sort of middleware service that can be used in order to deploy components on an active node. Therefore the clients do not need to implement themselves routines that discover/download the components and check their security credentials. The ACM's interfaces can be used instead, hence we have thinner, easier to implement and inexpensive to maintain clients. The instantiation of the component is not covered here, and is technology specific.

Optimized software distribution: The ACM is able to allow reuse of one component by many users instead of downloading many times and handling as different each component installation the user requires. Although the future networks are expected to provide high bandwidth, the mobile ones will still provide users with limited bandwidth allocation (at least until the 3G infrastructure is widely used) in order to optimize the resource usage.

Automatic Discovery/Indexing: The ACM is able to provide indexes of the code itself hosts or is available within a domain. It can query metadata enabled search engines or other ACMs in order to find the desired code. It can also build P2P networks of ACMs that exchange info and discover other components. Furthermore the ability to search for new code based on the profile inspection may prove interesting to the developers.

XML-based metadata: Instead of deploying application specific home-grown solutions to represent information about the components, ACM uses XML and RDF together with DCMI in order to provide interoperable widely accepted flexible profiles.

Security awareness: The ACM is able to deploy/use digitally signed XML profiles. The digital signature owner can be extracted and further used for authorization decisions. Furthermore it is also made sure that the resources that the profile refers to have not been altered. Interesting are also the implications of the XACL and the new capabilities it offers within the whole process which have not yet been adequately investigated.

Safety: The ACM indirectly takes care of the safety of the active network node. It makes sure that no untrusted or compromised code is installed. Although it cannot control on runtime the executed code, it can via policy permit only to the administrator to install code he has tested and is sure that runs safely for the systems configuration. Furthermore the upgrade functionality, although primitive, allows quick deployment of the latest version of the components that can be pushed to active network infrastructures once made available.

Openness/Dynamicity: The ACM by using promotes openness and supports on demand component downloading. Therefore it eases the tasks of future services and dynamic environments e.g. where the protocols of the active nodes that recently joined the network have to be adapted and the missing functionality to be fetched.

9 Conclusion

We have presented several directions that we consider as challenges for the component deployment context within next generation converged networks. Active networks, which are also seen as one of the key enabling technologies also for mobile networks [3], depend on code that has to be discovered, installed, deinstalled, indexed, distributed and profiled. The ACM as presented here tackles many of these problems in an open, scalable, robust and extensible way. We have presented the architecture of the ACM and its relation to other components such as the security manager and other ACMs.

One could consider that such a liberated dynamic framework for code distribution can be considered as a disaster recipe that will advance the expansion of viruses and relevant malicious software. However, this does not hold true. It has to be pointed out that currently users of open source software download updates via a well-known sites and their mirrors, which they have to trust. The approach presented here allows more sophisticated mechanisms to be deployed, since a generic policy-controlled framework is in place, which can partially enhance existing distributed code deployment approaches. As an example we have pointed out that P2P enabled ACMs can tackle secure on-demand patching to a great network segment proactively, therefore minimizing the overall vulnerability of the network. Of course the approach is very generic and the development was done only to prove that the concept is promising. However, the author believes that this might be a way that can lead us to autonomous patching which will result in more secure networks. Furthermore code-profiling can offer richer info upon which more fine-grained policy and trust decisions can be made by the end-user at different levels.

What this approach tries to promote is also the XML/RDF based profiles for unambiguous description of the components and better component discovery, a step closer to the fulfilment of the semantic web vision. Recently the W3C announced the Composite Capability/Preference Profiles (CC/PP) structure and vocabularies (http://www.w3.org/TR/CCPP-struct-vocab), which is definitely a step towards the right direction and complementary to our work. By adding security and embedding it to the ACM approach presented here we may be able to realize powerful interoperable code distribution and policy-enabled management of it. Finally we envision the future of ACM as a P2P network, possibly with Instant Messaging capable nodes, that will be able to take advantage of the underlying programmable infrastructure.

10 References

[1] Stamatis Karnouskos, "Security Implications of Implementing Active Network Infrastructures using Agent Technology", Special Issue on Active Networks and Services, Computer Networks Journal, Volume 36, Issue 1, pp 87-100, June 2001 (ISSN 1389-1286).

[2] Stamatis Karnouskos, "Realization of a Secure Active and Programmable Network Infrastructure via Mobile Agent Technology", Special Issue on Computational Intelligence in Telecommunications Networks, Computer Communications Journal, Volume 25, Issue 16, pp. 1465-1476, October 2002 (ISSN: 0140-3664).

[3] A. Fernando, D. Williams, A. Fekete, and R. Kummerfeld, "Dynamic Network Service Installation in an Active Network", Computer Networks, 36:35-48, 2001.

[4] Jonathan M. Smith, Scott M. Nettles, "Active Networking: One View of the Past, Present and Future", IEEE Transactions on Systems, Man, and Cybernetics (T-SMC), Volume 34, Number 1, pp. 4-18, February 2004 (ISSN: 1094-6977).

[5] Spyros Denazis, Stamatis Karnouskos, Toshiaki Suzuki, and Satoshi Yoshizawa, "Component-based Execution environments of Network Elements and a Protocol for their Configuration", IEEE Transactions on Systems, Man, and Cybernetics (T-SMC), Volume 34, Number 1, pp. 82-96, February 2004 (ISSN: 1094-6977).

[6] B. Blobel, P. Hoepner, R. Joop, S. Karnouskos, G. Kleinhuis and G. Stassinopoulos, "Using a privilege management infrastructure for secure web-based e-health applications", Computer Communications Journal, Volume 26, Issue 16, Pages 1863-1872, 15 October 2003 (ISSN: 0140-3664).

[7] Angelos D. Keromytis, "Patch on Demand Saves Even More Time?", IEEE Computer, pp. 94-96, Volume 37, Number 8, August 2004

[8] Stuart Staniford,Vern Paxson, Nicholas Weaver, "How to Own the Internet in Your Spare Time", Proc. of 11th USENIX Security Symposium, pp. 149-167, San Francisco, August 2002 (ISBN:1-931971-00-5).