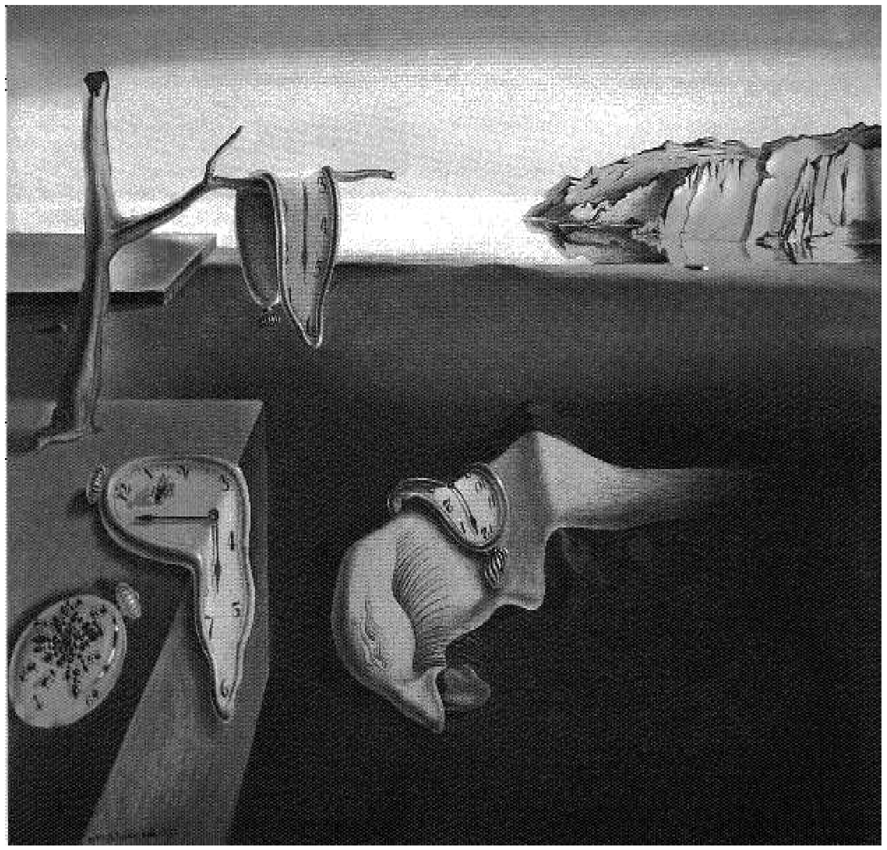
Institut für Automation
Abt. für Automatisierungssysteme

Technische Universität Wien

Projektbericht Nr. 183/1-111
July 2000

Security Issues in W2F

Bettina Weiss



Salvador Dali, "Die Beständigkeit der Erinnerung"

Security Issues in W2F

Bettina Weiss

July 2000

Contents

Abstract

This report is dedicated to the security policies required for our project W2F. As we will see, W2F is not yet in a state of development where we can give policies for everything. We will instead try to identify some of its needs and the areas where further work is required.

1 Introduction

In our project W2F[1] (Wireless/Wired Factory/Facility Fieldbus), we intend to develop a next-generation LAN/fieldbus which will properly address distribution, security, fault-tolerance, and real-time issues as well as flexibility w.r.t. wireline/wireless interconnections [FS00]. In the security field, we will have several decisions to make:

- What requirements to security do we have?
- Which security protocols should we implement?
- Should we use existing protocols or develop new ones?
- How can we prove that the protocols we choose are fault-free?

The following report will address these issues. Section 3 tries to create system models suitable for categorizing security protocols. Section 4 will give an overview on the W2F project and point out its security requirements. Section 5 lists the services provided by W2F. Section 6 identifies the components of W2F. In Section 7, our security protocols and their positions within the W2F protocol stack will be described. Finally, Section 8 will address open issues which need to be solved.

Why do we need security in the first place? Our system consists of n nodes which are connected over a network. It provides services for the user (e.g., process automation) and for some of its nodes (e.g., clock synchronization). Of course, we expect the system to accomplish its tasks reliably. This requires that the nodes which are providing a specific service are selected with care and behave as expected. These nodes will have to be checked somehow, probably off-line, and marked as capable of participating in the service. But even if the system is assembled and started correctly, we must enable our system to handle the dynamic exchange of components, either because they have failed or because of an upgrade. As soon as we allow such an exchange, however, we must also provide the systems with a means to detect malicious use of this feature. So we need a mechanism that allows members of a service group to assay the capabilities of a potential new group member and to ascertain that the aspirant is not malicious.

Apart from internal services, our system conducts tasks for its human user, who might want to keep its activities and data secret. Therefore, we must enable our system to keep sensitive data unaccessible to outsiders, both on the storage hosts and during communication. We will need some kind of access control to the hosts and encryption for communication.

[1]The W2F-project (*http://www.auto.tuwien.ac.at/Projects/W2F/*) has received funding from the Austrian START-programme Y41-MAT.

2 System Description

The W2F project targets automation (factory, home). So we have lots of different components (small sensors, servers, external PCs, ...).

On the lowest level, we assume a system consisting mainly of sensors and actors together with one controller (several controllers if we need fault tolerance). The components are grouped together because they are physically near each other and share a common goal/task. They will form a *cell*. We assume that a cell can have an upper bound on the number of its components.

Several cells can be connected to form a larger structure, which is unlimited in size. So our system consists of an unlimited number of cells, each of which combines a limited number of components.

An example (white circles are sensors, black circles are controllers):

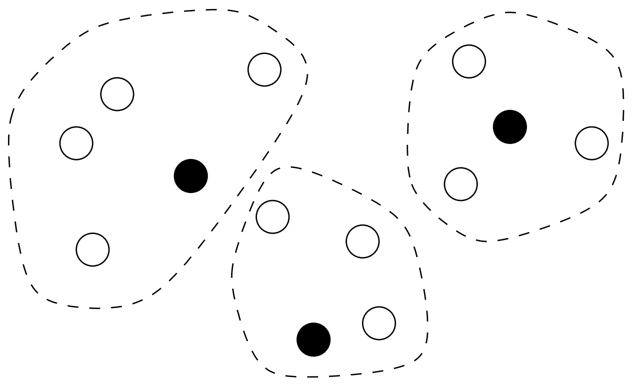


Figure 1: *Three Cells*

As one can see, the grouping of sensors to controllers has no direct connection to the location of the sensors (the controller does not have to be in the center of the cell and it does not serve all sensors nearest to it). The grouping is mainly functional and only loosely based on the location of the objects. The only real restriction is that wireless sensors will require a controller within their transmission range.

Imagine for example a tour guide system in a museum. In our case, the sensors used for locating the position of a visitor within one room together with a node responsible for controlling these sensors are forming a cell. We can savely assume that this small system will not need to scale well (larger rooms can be served by two such cells if necessary). The cells are connected to form the whole system. Since we do not know how many rooms the museum has, or if the exhibitions are distributed over several buildings, the system must scale well.

How do controllers know which sensors they serve? Obviously, the members of a cell should form a group. If we use secure group communication techniques, then we can even ensure that messages from a particular sensor can only be decoded within its group, regardless of who else is within sending range.

What should we do with new sensors? How do they know which cell to join, if cell boundaries have nothing to do with location? Possible solution: put in the sensor, let it broadcast a join request, but notify the intended controller that it should react to the request. All other controllers will ignore the request. This is not bad because it makes certain that you cannot uncontrollably add sensors to a cell. Every new component is "authorized" by the system. This does not make the system less flexible (a human has to sit down and notify the controller vs. fully automatic join – since the sensor has to be installed anyway, one additional action is no loss of flexibility), it just adds a little bit more security to it. When the sensor has joined (and its credibility has been checked), it gets the group data.

But of course, this is not very fault tolerant. If a controller breaks down, its sensors should be allocated to other controllers. This should happen dynamically, without human help. And in any case, it is rather insecure to let a human decide on the controller. We need some way to make sure a new sensor finds the controller it should work with. But how? What feature decides to which cell a sensor belongs? In the museum example, it is the location. The controller has the task to track human movement within the room, and every sensor in that room adds to the data used by the controller. So the controller has a given 3D-range for which it is responsible, and the sensor simply is in this range. It would suffice if the sensor were aware of its location and would send it in the initial cell joining phase.

What other possibilities are there? Functional reasons. Perhaps one controller in the room is responsible for movement tracking, another for controlling the room temperature. In this case, the tracking sensor is allocated to the tracking controller and not to the temperature controller. So it is the location as well as the functional specification which controls the allocation. What if we have two controllers doing the same things? Both could use the sensor, or they could decide on who gets the sensor.

3 System Models

Although there have been many proposals for security protocols, most of them do not explicitly state any system assumptions. This commonly leads to misconceptions about the protocols.

Inspired by the sync./async. model of distributed systems, we will try to discern several system models suitable for security protocols. The models can be used to position a protocol and to clarify what powers an adversary can have.

Currently, attacks on protocols are launched with an all-powerfull adversary and solutions are suggested, but obviously nobody has ever bothered to state that a protocol is suited for one environment but not for another. For example, many attacks assume that the intruder has complete control over the network and can remove and insert messages at will. This may well be the case in a wired network, but might be impossible in a wireless environment. So we need a system to categorize protocols into those that work only in the wireless world and those that work in wired systems as well. The models should be hierarchical, so we can, e.g., show that a protocol working in the wired network will also work in the wireless network. Other things the models should include are the capabilities of the principals (sync. clock, local clock, amount of computing power, ...), of CAs, ...

4 Security Requirements

We expect that even sensor data may have to be encrypted, and sensors will need some credentials to participate in the network. Sensors must have certificates and at least be able to sign their data. Therefore, the interface between the sensor and the CDMA-network, which features the CDMA-sender (and perhaps a receiver), must also contain basic security features. Nevertheless, the interface must be cheap. (These might be mutually exclusive goals...)

Sensor data must always be signed (so we can check if the data comes from a good source) and perhaps be encrypted (if the data is sensitive). A signature requires some random text in the message to avoid replay-attacks. This could either be a random number or a timestamp.

Actors have at least a CDMA-receiver. They must be able to verify the origin of control commands (check a signature).

Units which process the sensor data and provide user data are called servers (S). They may or may not have a user interface for login. Servers are part of the W$_2$F system and all software comes from a known and trusted source. Servers must have the facility to check the source of data (verify signatures). They must also be capable to sign control commands. They might get data access requests from workstations, so they must be able to verify the access rights of users.

Since a company will also have lots of PCs with common operating systems like Linux, Windows, Mac-OS, ..., which are used to generate reports and such things, these computers should be allowed to gain access to our system to gather data for status reports. They should only have read access. We term such computers with foreign and untrusted operating systems workstations (WS). They must be able to offer users secure login.

We will need authentication and secret key exchange (perhaps public keys?) at very low levels in the protocol stack. We might implement other security issues on higher levels, when synchronized clocks and other services are available. But for the low levels, nothing except the existence of unreliable datagram services can be assumed. We should also assume that an intruder has control over the network and can replace messages with his own (this has the advantage that the network card need not be part of the trusted computing base).

5 Service Groups

Services are generally provided by a group of nodes (for fault-tolerance reasons). A node gets some kind of certificate stating which services it can provide. It can then join services if necessary. Our system should be able to optimize service groups so that the load on the nodes is balanced (otherwise, a very good server might join every group, get overloaded, and in the end reduce the quality of the service). Fault-tolerant and reliable group management is very important in W$_2$F, it is one of the core problems we have to solve.

Should a super-user (SU) be able to set up the system, i.e., specify the members of service groups? Definitly not. The system should handle such things automatically. Ideally, human intervention should be kept to a minimum because it is a security leak. The SU should be able to remove a node from a group (because it is faulty and the system did not notice that), but should not be able to force a node into a group. The system is responsible for assessing the capabilities of the node and only the system can allow a node to join a group. This removes the problem of an SU undermining a service by adding malicious nodes and removing the good ones. What if the SU removes all nodes from the group, i.e., makes the service unavailable? The group must always know how many members there are, or at least notice when a given lower limit necessary for keeping up the service is reached. At this point, the group should notify the system and start enlarging the group.

So the system automatically constructs the service group and manages the membership. It joins and removes nodes as required. A human user may remove nodes from the group but may not add new nodes.

What should our (distributed) group management algorithm be able to do?

- Create a group.
- Keep track of the number of members. Perhaps keep track of the number of good members (dynamically determine the number of faulty members!).
- Join a new member.
- Remove an existing member.
- Merge a group.
- Detect splitting of a group (network partitioning). Automatically done if the number of good members is determined dynamically.

After starting the system, service groups begin to form. When a node is started up, it sends its service certificates to all service groups it can participate in and offers its own services. Three reactions are possible:

- The group does not exist. The node will experience a timeout and assume that it has to form the service group. It will enter itself into the group and broadcast a join request for the service to all nodes to get new members.
- The group exists, the participation offering is denied. The node is not part of the service. It should nevertheless poll the service periodically to avoid that the service is lost (due to some error or because of network partitioning).
- The group exists, the offering is accepted. The node is now a member of the group.

When a group determines that it has too few good members, then it will broadcast a join request. Nodes capable of participating will reply to the request and the group will elect the new member.

When a group determines that it has too many members, it will elect one node and send it a leave request. The node then stops participating in the service.

We will need some algorithm to determine which node is the best candidate for joining a service. Every candidate will be evaluated (also against the nodes already in the group) and will be selected if it is the best candidate. It might replace an existing node if the existing node provides worse service.

If two groups providing the same service have formed (due to network partitioning) and the groups detect each other, then the groups will merge into one large group and this new group will remove any surplus members.

Since services might have to use other services, each member of a service group should get a certificate stating that it does provide a certain service (in addition to the certificate stating that it is capable of providing this service). This is the node's ticket for using other services. Of course, passive usage does not require a service certificate (like listening in on clock synchronization messages), only active usage (like data requests) does require such a certificate.

All group action (join, leave) is logged. Suspicious actions can at least be detected by a human user, or perhaps by some automated tool (perhaps even an artificial intelligence program?).

6 Components

Abbreviations used in the following sections:

S: server

SU: super-user

system: the W2F system

WS: workstation

6.1 Workstations

The workstation (WS) is the interface between the company and the system. It uses an unknown operating system and must be considered as insecure. Two scenarios are possible:

1. Access to W2F is gained through login from the WS.
2. Access to W2F is gained through a web-server.

In the first scenario, the WS is equipped with a network interface card (PCI slot or similar) and its driver. The card might be able to verify the integrity of the driver, or perhaps the card must be able to load and install its own driver (if such a thing is possible). Either the card or the driver must know that the host is not secure and notify the system through a low-level certificate or something. Even a super-user (SU), when logging onto a WS, only has read access to the system data.

In the second scenario, the data is gained by browsing and no particular access restriction on the WS is necessary.

If we do need some access feature, then the WS must provide the user with a secure login protocol, secure data access, and reliable data synchronization. A session consists of three phases:

1. Login
2. Work
3. Logout

In the first phase, the user is prompted to enter a login-name and a password. The station then tries to access the system and download the user data (work profile). The login phase can result in one of the following outcomes:

- The WS detects that the system in general is currently unavailable, perhaps due to a communication problem. The user is notified and logged out. If the WS checks the availability of the system periodically, such a breakdown could already be detected before the user logs in and a message could be shown that login is currently useless. However, we might not want an outsider to see on the screen whether the system is available from a particular WS or not. A good way to check general availability might be for the system to ping its interface nodes from time to time. The WS expects the periodic ping and notifies the user on screen if it misses a fixed number of pings. Another way of checking could be the clock synchronization service. The WS will want to use this service as a client and will know when the service is not available. It can then notify the user on screen.
- The WS detects that the authentication service is unavailable. The user is notified and logged out.
- The WS gets an authentication-failed message from the authentication service. The user is notified and logged out.
- The WS detects that the database service is unavailable. Since this service provides all user-relevant data, the user cannot use the WS without it. So the user is notified and logged out.
- Authentication succeeds and the user data is available. The WS downloads the user profile (work profile like desktop layout, ...) and presents the desktop.

In the second phase, the user accesses data (system status, personal system messages, ...) for further processing (in reports).

We also have to decide whether we keep a local copy of the data in the WS for performance reasons. If we do, we have to guarantee that the data remains secure. We might also need a database service which handles data synchronization.

In the third phase, the user is logged out. Any data not synchronized until then is synchronized now. If the data cannot be sent to the system, the user is notified and can either discard the modifications or wait until the system gets online again. Since it would be a pain to discard many changes, keeping the WS synchronized with the system database would ameliorate such communication problems, because the user is notified at once.

6.2 Servers

The server (S) is a powerful node which is able to provide services and maintain critical data. It is controlled by the W$_2$F operating system. A malicious server should not be able to weaken the overall system security, so we might not have too many requirements to the security of the system. However, we must ensure that good servers can detect a malicious server.

Concerning the user interface: our login program allows access to the system, but not to the machine which is providing this access. Only a system administrator with access rights for the machine can modify the machine itself. The machines might be classified according to the services they provide, and only a sysadmin with the proper rights can modify a machine of a given class. So we need to classify users and sysadmins as well. Each user has a set of personal access rights to every component or every component class in the system. We need a good distributed database to manage these sets.

6.3 Users

Although strictly speaking not part of our system, users play an important role in the security of the system. Since every user gets a personal set of access rights, the selection of these sets is very sensitive and can open security holes.

We should provide a couple of commonly known fixed roles (Super-User, Administrator, User, Technician, ...) to guide companies in the choice of sensible sets. The final choice of access rights is up to the company and may be a security hazard. Either we design our system in such a way that even a bunch of malicious super-users cannot destroy it, or we have to provide a guideline which assesses the damage done by assigning one or more malicious users a given set of access rights.

I also recommend using at least two persons for every major system change. Perhaps major changes (like adding/removing nodes, restructuring service groups, ...) should be automatically broadcasted to a whole group of sysadmins, apart from being logged? A log file may be edited by a super-user, but not even a super-user should be able to modify personal data of other users. Each user should get a private space with personal data (the things which are downloaded when the user logs on), including a message pad where everybody can send to, but only the user in person can read/delete. Of course, the above-mentioned log file could be implemented in such a way, as some kind of personal message pad of the system, and anybody with the correct access rights can read the messages.

If we require two persons (accounts) for important changes, than we have to ensure that no single person uses two accounts. To make the four-eyes-principle really secure, we would have to use biometrical methods to determine the identity of the users. If we cannot do this, then we have no means to check whether a malicious super-user creates for himself the two accounts needed to perform an action.

Do not forget: users logging in from a WS cannot be protected from malicious software. In Windows, e.g., every keyboard input can be caught and stored by hook functions before they are passed on to the application software. These hook functions can be installed by any program, a fatal situation for a login program. Although the functions can be removed by the login software, other (undocumented) ways to protocol keyboard input may exist. And of course the keyboard itself might be rigged (a commercial hardware solution does exist). So users should not use critical passwords when accessing the system from a WS.

A biometric input device would also circumvent the problems of entering a password on a hostile machine. On the other hand, we would need a secure line from the input device to our system.

We already mentioned that the SU can remove nodes from a group. It might be a good idea to use certificates for these actions. What I would like is a certificate template which can be filled out with the name of its holder and a certain action. When the account for the SU is created, a certificate is generated which contains the name of the account and the remove-from-group action. The certificate can be verified and the SU has to provide it together with a proof of authenticity (signature) to really remove a node from a group.

The important thing is: the certificate is a template and is filled out dynamically. So we do not have one statically generated certificate that allows to remove a member from a group, but we generate such a certificate for every person that is allowed to perform the action. Thus, a certain account can loose trust and get its certificate revoked without revoking the certificates of all other persons.

Instead of the certificate, we can also use a database service to store access rights. In this case, the system checks the database before executing an action, i.e., we have three messages (action request, check, reply to check) before an action is performed. So it is better if the node already sends a valid certificate with the action request. To avoid problems with revokation, the certificate could be valid for a short time (this requires a trusted time base!).

7 Security Policy

In this section, we will describe what requirements to our security policy exist and how we might be able to solve certain problems. See the technical report [Wei00] for an overview on commonly used security protocols.

7.1 Authentication

We can use an asymmetric encryption scheme and use the private key to sign messages by encrypting the hash of the message. The receiver can then verify the sender's identity by applying the public key to extract the hash and comparing it to its own hash of the message. This also ensures the integrity of the message. In order to avoid replay attacks, messages should include a timestamp or some other freshness indicator.

7.2 Key Distribution

If we use an asymmetric encryption scheme, then information about current keys is exchanged with *public key certificates* and *revocation lists*. The former are used to assign public keys to users, the latter is necessary if the public key has to be changed. If Alice has originally been equipped with the key K_{A1} but has changed it to K_{A2}, then the revocation list will contain the certificate for K_{A1}, a certificate for the transfer from K_{A1} to K_{A2} signed by the certification authority (CA), and the certificate for K_{A2}. *[Perhaps the list need not contain all that information. Its purpose is to assure everyone that K_{A2} is Alice's current key and that K_{A1} should not be used anymore.]*

The problems associated with public key encryption are:

1. Getting the first public/private key pair.
2. Changing the key after if has been compromised.
3. Getting the most recent revocation list of another user.

I believe that many problems can be solved more easily if we assume that Alice and the CA share a common key $K_{A,CA}$ which is known only to them. The key is only used when Alice requires a key pair and should be kept in a highly secure environment. If this secret is compromised, then Alice cannot participate in the system until she and the CA get a new secret (entered manually). The CA should also have a public key which cannot be compromised. If it is, then the whole system breaks down and has to be reconfigured manually. So our requirements to the system are

- A certification authority (CA) exists.
- The CA has a public key which cannot be compromised.
- The CA does not disclose its private key.
- Each client shares a secret key with the CA (the CA needs to store a key for every client).
- The secret key cannot be compromised (is implied because it is used very seldom).
- The client does not disclose the secret key (needs special hardware for storage).
- The CA does not disclose the secret key.
- The CA can generate good public/private keys.

After being installed in the system, Alice uses the secret key to request a key pair. The CA sends it to her. From then on, Alice only uses the key pair for communication, so disclosure of the secret key is unlikely.

1. A $\rightarrow$ CA: $\{\text{Key Request}\}_{K_{A,CA}}$
2. CA $\rightarrow$ A: $\{\mathrm{K}_A, \mathrm{K}_A^{-1}, \mathrm{Cert}(\mathrm{K}_A)\}_{K_{A,CA}}$

I would suggest the following protocol for getting the most recent certificate. In the protocol, Alice wants to talk to Bob and needs his current certificate. She only trusts the certification authority, so she asks the CA for the most recent list. To avoid overhead, and also for security reasons, each list has a unique identifier (counter), so a revocation list is uniquely identified by the pair $ID_B = \{\text{Name, ID}\}$.

7.3 Communication

All data should be encrypted before it is sent. This includes the header (address field) as well. The encryption should be done by hardware and use a CBC mode to avoid substitutions. Each network controller encrypts the message and sends it on the bus. Incoming messages are decrypted with the secret key of the network card and the address field is checked. If it is addressed to the client, then the rest of the message is decrypted and relayed to the higher protocol layers. Note that a symmetric encryption is less reliable, because all keys must be stored in the network card or the card must at least be trusted. In a pulic key system, the public key of the receiver is common knowledge and can be stored by the application. It is handed down to the network card together with the message. Only the secret key associated with the card (and its client?! – might be a problem?) has to be stored in the card.

How is routing handled? We could make a public key for routers and send the encrypted message plus the address field encrypted with the routing key. The router can keep a table of such encrypted address fields and forward messages accordingly. So a message to Alice has the form ($\{A, M\}_{K_A}$, $\{A\}_{K_R}$). A problem occurs if an intruder can control the network, because he could substitute $\{A\}_{K_R}$ with $\{B\}_{K_R}$. This does not matter if the message does not have to be routed, but is a problem if routing is thus suppressed.

We do not yet know how to solve this problem in a secure way. Signing of messages is impossible because we want to hide both sender and receiver. We need some way to make the two parts inseparable (or at least tamper-aware) without giving away any identities.

8 Open Issues

To begin with, we need a good system model. Up to now, apparently nobody cared about it, but I think that it is required if we want to compare algorithms fairly. It also helps to define one's needs if one can read about different models and then decide which fits best.

Then, we have to decide what our system is capable of and what it needs from security. For example, we have sensors. What do we want to do with them? Is their data confidential? Do actors need to authenticate the sender of commands? Do these components need to execute protocols for group management/participation? What about the computers controlling them? What applications do we expect in the first place? It may very well be possible that different applications of W2F need different security solutions. We have to identify what we are catering for.

When it is clear what our target applications are, we have to do a list of requirements for each possible application. From this list, we can identify the services we should provide. The level of security integrated into these services may well be a different one for different applications.

Finally, when we know what we need, we have to find ways of doing it. This is the stage where we decide what protocols to use. It is also the stage where we develop the general security framework (decide whether we need logging capabilities, decide about access policies a.s.o.).

Finally, we have to verify the correctness of our protocols and framework. There are many tools for protocol verification, but I have not encountered any for framework verification up to now. Maybe we will have to develop our own verification mechanisms for that.

References

[FS00] Christof Fetzer and Ulrich Schmid. Architecture and services of the W2F fieldbus. Technical Report 183/1-101, Department of Automation, Vienna University of Technology, (forthcoming) 2000.

[Wei00] Bettina Weiss. Security in distributed systems - a survey. Technical Report 183/1-99, Department of Automation, Vienna University of Technology, February 2000.