MIT Communication Core

Issues to consider in the design

GRUPO **AIA**

Rome meeting, Feb 8, 2007

Overview

- These are just a set of general thoughts (and some particular notes) about the issues arising in the technical design of the **MIT Core**, i.e. the **Commmunications Layer**.
- The main point of view we want to contribute here is that of the **Independent Software Vendor** (ISV): the people who provide LCCI with software systems and integration.
- Keep in mind that the actual success or failure of many new proposed standards depends ultimately on the rate of acceptance and adoption by ISV's.
- In this talk, we'll try to build our case:
  - *Unless you don't mind that the MIT technologies fall into irrelevance a few years down the road, try to design something **simple**, small enough to be **comprehensible**, and make an extra effort to ease the **adoption** by ISV integrators.*

Contents

1) Communication modes

2) Communication technologies

3) Security technologies

4) XML schema design issues

Communication modes

- The initial draft design is based on the exchange of XML files. This has many good points:
  - At the higher-level (semantics), the file becomes the "quantum" of information being exchanged, which is quite good for traceability, auditability, and debugging.
  - Decouples the transport mechanism from the content. This, for instance, allows encryption to be applied not only on the channel, but on the "persisted" content (i.e. the XML files).
- However the downside is that now we have to say **how** we intend to interchange those files.
- Which means that, irrespective of the lower details of the transport protocols (most likely TCP), we have to specify two things:
  - the **modes** of communication (i.e. push/pull, stateless/stateful dialog, etc)
  - the **protocols** to support those modes

Communication modes (II)

- Initially, the draft suggests three types of messages:
  - Informational
  - Negotiation
  - Administrative
- Our interpretation of the intended design, in terms of communication modes & protocols would be summarized as follows:
  - The main communication modes between the parties (the Service Provider and the Service Consumer) are both “pull” (request--> response) and “push” (under subscription).
  - Additionally, there's a need for “negotiation” dialogs between the parties
- We make this distinction because from the point of view of the technical implementation, these two classes of communication are very different:
  - The push or pull messages constitute what we would call a completely “state-less” communication
  - The negotiation messages are part of a dialog, and therefore this needs the specification of a **protocol**.

Communication modes (III)

- Obviously, state-less communications are to be preferred when possible
- Stateful (dialog) communications need a carefully designed protocol: a "state-machine"-based modelling that should be as simple as possible, and which takes into account all possible failures in the middle of the dialog.
- So therefore, we roughly have:
  - Informational files: they're part of push or pull comms; no protocol needed because there isn't really any dialog.
  - Negotiation: some protocol definitely needed
  - Administrative: some protocol needed, if only just for ACK/REJECT

Warning, complexity ahead: aim to simplify the rules for the dialog (i.e. the **protocol**) that takes place during the exchange of "ServiceNegotiation" messages

Comm technologies

- Clarifying the previous design issues is also useful because it lets us think how we may use **existing technologies to layer upon**.
- We mentioned how it is a good idea to base the communication on XML files and be agnostic about the transport mechanism.
- Years ago, the EDI people had to specify a transport standard (VAN & AS2). Luckily things have changed, the whole world has standardized on Internet protocols.
- At the lower level of the transport, TCP is the most likely candidate for actual implementations.
- At a higher level, we need at least a network protocol that handles all the "systems stuff" needed to transfer files. Most visible candidates are:
  - **FTP/SFTP:** the classical way to transfer files. Will handle push/pull if we have an FTP server at both ends.
  - **Web server (HTTP):** at a functional level, equivalent to serving files via FTP, but push mode is more awkward.
  - **Web services:** a more flexible way to interchange XML files, but more complex. It was designed with the paradigm of RPC (remote procedure call) in mind, not file transfer per se.
  - **Email (SMTP):** a slow and asynchronous (but quite reliable) way to transfer files. Pull mode can be implemented with little effort.

Comm technologies (II)

Note: none of the obvious existing protocols will help much with our MIT-specific stateful type of communication. Most likely, we will have to design and ad-hoc protocol on top of some "dumber" protocol like FTP or web-services.

- This is for instance what web applications do: they build their own state management (web session management) because HTTP doesn't do it for them. This management is very specific, not only to the programming language (eg. Java vs PHP), but to the application!

General thoughts: design simple (MIT-dialog-)protocols and provide a reference implementation atop some widely known transport protocol. Make it easy for future implementors to build interoperable "MIT server end-points".

Security technologies

- If there's an area where we should **never** try to reinvent any wheels, that is **security** (or, more specifically to our case, encryption and authentication). There are tons of high-quality, strong-encryption products which have become COTS.
- The roadmap seems more or less clear:
  - For the XML files, **mandate** the use of existing standards (W3C being the most likely ones) for encryption and digital signatures.
  - For the transport, only **suggest** the use of existing standards like SSL/TLS or OpenSSH, as an additional layer of protection.

Warning: the really hard part about security is not the tech, but the human factor. Specifically, key management and chains of trust are always problematic. Who will be the CA?

XML schema design issues

- XML format as a data container is generally speaking a very good idea these days
- But watch out for **over-engineering** and the "kitchen-sink" syndrome creeping up in the XML world. Avoid fads, try to assess the real value of the standards for our Project, on a case by case basis.
- Case in point: the use of W3C Schemas. In XML parlance, a "schema language" is the system in which you define the tags, structure, etc. of the XML files you want to use in your Domain. There are three main (competing) schema languages to choose from:
  - DTD (considered old)
  - W3C Schemas
  - RELAX NG Schemas
- Many experts consider W3C Schemas to be overly complex and advocate the use of RELAXNG instead.
- Hopefully our MIT schemas do not need to be so complex that the use of W3C or RELAX makes any difference. But the point is, let's not just follow fads in IT.

XML schema design issues (II)

- The first rule of XML Schema design is *"don't do it unless you really need to"* (Tim Bray)
- The bad news is that IRRIIS is tackling a genuinely new domain, in which there is not any standardization
- Other domains, like business-to-business in manufacturing industries or retail distribution, have plenty of them: EDIFACT / ANSI X-12 in the past, ebXML and UBL in the future.

- Ejemplo EDIFACT:

```
UNB+UNOA:1+ENSENADAINTERNATIONALTERMINAL+TRANSPORTACION+010717:0755+258+++UNH+2572+BAPLIE:
1:911:UN:SMDG15BGM++2572+9'TDT+20+003WB++:103::TMMSINALOA++MLL:172:20'LOC+61+MXZLO'DTM+178:
0107170755:201DTM+133:0107170755:201
```

- Ejemplo EDI-XML:

```
<?xml version="1.0">
        <document>
                <purchaseOrder>Orden de Compra</purchaseOrder>
                        <orderNumber>1234556</orderNumber>
                        ..............................
        </document>
```

XML schema design issues (III)

- The design of the **MIT XML Schema** will be probably the most relevant contribution of this part of the IRRIIS project. After all, it could be argued that all the other technical issues are straightforward by comparison.
- The bad news is, most experts in XML language design will tell you that getting it right is very hard.
- It's mainly a problem of **semantics** and **modeling**: it will be difficult to avoid some semantics gaps
- So in our case it's doubly hard because you have to get people from **different sectors** and have them agree on semantics
- Since this is hard, I advocate for a minimalistic approach, and build iteratively. It will be difficult to really understand all the issues involved before we actually gain experience with some actual implementation.
- Other considerations: plan for **extensibility**. Apply the **MustIgnore** and **MustUnderstand** principles in order to accommodate future extensions.
- A related engineering principle might be useful, since we want **resilient** MIT communications: *be strict with what you send, be liberal with what you accept* (from the creators of SMTP). In other words, mandate XML strict **validation** when sending, but not so strict when receiving.


GRUPO AIA
Aplicaciones en Informática Avanzada, S.A.
Barcelona
Artesans 10
Parc Tecnològic del Vallès
08290 Cerdanyola, Barcelona
Tel 34 93 504 49 00
Fax 34 93 580 21 88
Madrid
Núñez de Balboa 35 A, 2º C
28001 Madrid
Tel 34 91 576 23 27
Fax 34 91 576 13 02
California/USA
Group AIA, Inc
100 Bush St. Suite 1700
San Francisco, CA 94104-3919
Tel 415 981 70 00
Fax 415 772 82 34
www.aia.es