Consensus Software

Robustness and Social Good

Michael N. Huhns • *University of South Carolina* • *huhns@sc.edu*

Historically, a society is controlled by its systems of law, economics, and politics. Increasingly, modern societies are being controlled by computer systems as well, and it is not always obvious to which political philosophy such systems might adhere. Moreover, there is a danger if the systems contain errors and do not behave correctly.

In this column I explore some far-reaching issues of software development that lie at the intersection of robust software and sociopolitical systems. These two areas might seem unrelated — and most software developers would likely be horrified to have politics intrude on their programming efforts — but the intersection occurs through these premises:

- Software systems administer and control much of our societal infrastructure.
- People would appreciate and better accept that control if they had input into the nature of the control and the systems' behavior.
- Designers can make software systems more robust through redundancy, in which different versions of software components might cover for each other's mistakes and limitations.
- If many people could contribute software to societal control systems, the systems might be more robust and better represent people's interests.

The need for redundancy and the need for widespread participation can be mutually satisfying.

Societal Software Systems

The kinds of social systems I mean are exemplified by those for are electricity production and auction reputation. Running generators at a constant speed can optimize electricity production for efficiency, but this cannot realistically occur because the demand for electricity is not constant. To even out demand over time, consumers can defer nonessential electricity use to times when the demand is less. For example, a consumer could choose to run the clothes dryer at night when there is less industrial need for power.

Writing a centralized control algorithm to manage power allocation to thousands of consumers' diverse electrical devices would be extremely difficult and would no doubt annoy those who needed to dry their clothes and would not be allowed to do so. However, with incentives in place for using electricity at off-peak times and penalties for using it during high demand, individuals could provide a type of distribution control that would be more accurate; moreover, consumers would be happier because the system would work the way they wanted. In another example, when people buy and sell items via auction sites such as eBay, they develop reputations from the compiled opinions of people they have dealt with through the sites. This is a robust measure because fair opinions will eventually overcome any unfair ones. Interestingly, the users (rather than the software developers) are responsible for the reputation system's robustness: the greater the user community's participation, the greater the ratings' accuracy.

A Research Agenda for Robust Societal Software Systems

The above examples offer an admittedly utopian view of robust software systems whose behavior is a consensus of its clients' and users' wishes rather than a reflection of a software developer's personal bias. To realize that vision, we must answer the following questions:

 1089-7801/03/$17.00©2003 IEEE Published by the IEEE Computer Society

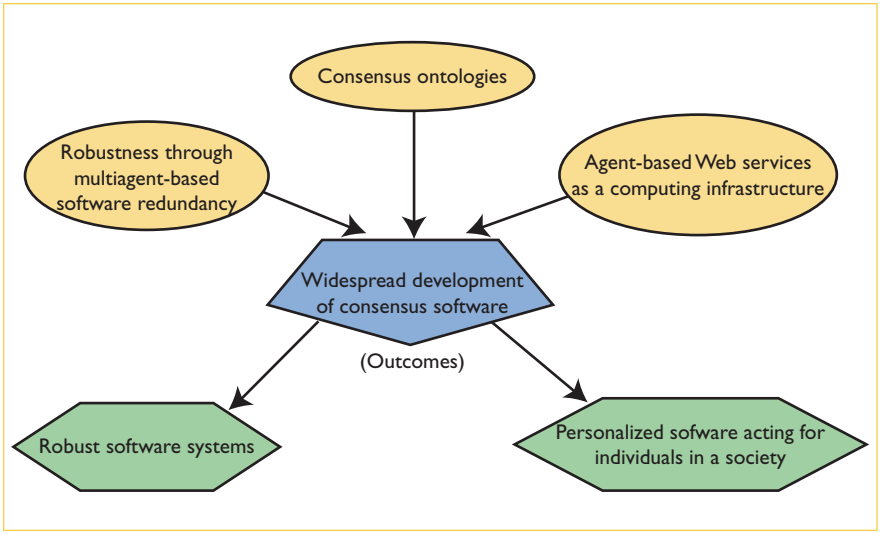

Figure 1. Necessary research threads. Widespread development can produce correct consensus software by taking advantage of research underway in multiagent-based software redundancy, agent-based Web services, and consensus ontologies.

- How can a wider range of people participate in software development and customization?
- How can we combine individual components with faults and errors into robust software systems?
- How can behavior be shared in a manner similar to how the Web enables information sharing?
- How can independently constructed autonomous components reach mutual understanding and consensus?
- What categories of software are suitable for consensus programming?
- How can we analyze and validate consensus software?

For guidance into the form that a consensus software system should take, we can look to both programming models and political models.[1] With object-oriented programming models, computer software resembles and represents the real world more closely than non-OOP paradigms, such as procedural programming methodologies, which typically represent mathematical abstractions. (For example, a Fortran subroutine might compute a matrix inverse, whereas an OOP module could represent the class `Automobile`.)

Although object-based and, better, agent-based software mimics the world, it is not developed in this way. That is, a small number of professional developers — not the "world" — produce the software. When I visit Amazon.com, for example, an agent with a model of me, based on what I look at or have bought previously, suggests items I might want to purchase. Amazon's software developers produce the model and, even though it supposedly represents me, I have no input in its configuration. Most distressingly, if the model inaccurately represents my interests and preferences, there is no way I can correct it.

As another example, my employer has a software and information model of me as an instance of an `Employee` class, but I did not contribute directly or consciously to its construction. Instead, all `Employee` models in my organization were constructed centrally.

Software Development Organized Politically

There is always a tension between the order that comes with centralization and the freedom that comes with decentralization. This is reflected in similar debates concerning privacy versus security, prevention versus protection, and, more generally, free markets versus centralized economies.

Economic historian Douglass North believed that institutions evolve toward free markets, in which institutions are conceptual structures that coordinate people's activities.[2] In his view, institutions comprise networks of relationships, including all the skills, strategies, and norms that the participants contribute. He believed the driving force behind the evolution of institutions was self-interest.

During the current economic downturn, many organizations are facing declining revenues and budget cuts. A common, centralized response is an across-the-board cut in salaries and expenses, based on the view that each employee should receive a minimum salary. Conversely, employees' individual self-interested views are that they should receive the maximum salary and that the company should make up the deficit elsewhere. Extending this view to all employees could lead to the organization's failure, and then no one would receive any salary at all. But the desire for increased salaries results in a pressure for institutional growth, and in this way everyone might eventually receive more.

In contrast, economist John Commons viewed an institution as a set of working rules that govern an individual's behavior.[3] The rules codify culture and practices and are defined by collective bargaining. As a result, institutions evolve ideally toward democracy. According to Commons' view, each `Employee` object in this example would let its salary reflect the organization's budget and the employee's particular contribution to the organization's performance.

Specifying Preferences

As societies attempt to coordinate and control their members' use of utilities and resources, individuals should have means to influence the coordination and control according to their preferences. Decisions can be made centrally or collectively, but individuals would benefit from active systems that could intercede on their behalf when dealing with dynamic systems that

require real-time decisions.[4] Examples of such institutions include banking, distributing electricity, determining routes for new roads, controlling traffic, managing telecommunication networks, and designing buildings or cities. Such societal services are beyond personal services.

When individuals contribute to a more accurate model of themselves and a more accurate characterization of how the systems they use should behave, the society will be easier to understand, more efficient, more productive, and more satisfying, and its principles will be better adhered to. People will be less likely to subvert a societal principle, such as conserve electricity, if they helped formulate the principle.

A Possible Solution

Although there are formidable research challenges in enabling consensus-based social systems, many of the necessary pieces are already in place. Figure 1 shows some of the necessary research threads in consensus ontologies and multiagent-based software redundancy and Web services.

First, I suggest using large-scale systems of computational agents that interact to achieve individual performance objectives and compensate for others' mistakes or limitations. Agents, unlike typical software components, can become aware of other agents via communication and interaction and have the potential to achieve coherent cooperative behavior. Just as multiagent-based systems have successfully combined information from disparate information systems, so too can they stitch together disparate software components' behaviors. More to the point, I have previously described in this column how multiple redundant agents can produce increased system robustness.[5]

Second, the Internet and World Wide Web have made it possible for individuals from different societies and cultures to share information. The Semantic Web will make it possible for computers to share not only information, but also functionality. It, and especially the forthcoming Web services, will let functionality be widely distributed, even when combined into coherent systems. Extending these results can provide ways for individuals to share behavior.

Third, to effectively interoperate, individually contributed components must interact, resolve conflicts, achieve mutual understanding and coordination, and behave as intended. Previous research[6] has shown that a multiplicity of ontology fragments, representing the semantics of independent components, can be related to each other automatically (without using a global ontology). This is possible through a *semantic bridge* –consisting of many other previously unrelated ontologies, even when there is no way to determine direct relationships among them. The relationships among the ontology fragments indicate the relationships among the components the fragments represent, enabling the information the components provide to be organized and understood. My research team conducted an investigation of the semantic bridge by relating small, independently-developed ontologies for several domains. A nice feature of our approach is that common parts of the ontologies reinforce each other, while unique parts are deemphasized. The result is a *consensus* ontology.

Finally, there is a relationship that must be explicated between the software systems that manage and control societal institutions, those responsible for setting up and managing those systems, and members of society. This relationship is similar to that in a publicly owned company, in which shareholders vote to elect a board of directors and decide on high-level corporate policies that employees enact.

Conclusion

Consensus software, as I define it here, can provide both a solution and an opportunity. It can help solve the common problem of incorrect software by improving robustness, and it can engender individual software customization and personalization. That is, it will enable a much wider range of people to develop customized software. This can lead to greater involvement by a broader cross-section of society and therefore greater acceptance of technology. It will help deliver on the Internet's promise of decentralization with its concomitant immunity to censorship, monopoly, and unequal opportunity.

By combining on-going research in consensus ontologies, agent-based Web services, societies of multiple software agents, and multiagent-based software redundancy, the research suggested herein would generate consensus software. The resulting software systems would be more likely to behave as people want and expect. Users would then more readily accept the coordination that the systems produce and exert. It's a revolutionary approach, but it can result in revolutionary improvements.

Acknowledgment

The US National Science Foundation supports this work under grant no. IIS-0083362.

References

1. P.E. Agre, "P2P and the Promise of Internet Equality," *Comm. ACM*, vol. 46, no. 2, Feb. 2003, pp. 39–42.
2. D.C. North, Institutions, *Institutional Change, and Economic Performance*, Cambridge Univ. Press, 1990.
3. J.R. Commons, *Institutional Economics: Its Place in Political Economy*, Univ. of Wisconsin Press, 1934.
4. M.N. Huhns, V.T. Holderfield, and R.L. Zavala Gutierrez "Achieving Software Robustness Via Large-Scale Multiagent Systems," *Software Eng. for Large-Scale Multi-Agent Systems*, LNCS 2603, A. Garcia et al., eds., Springer-Verlag, 2003.
5. M.N. Huhns and V.T. Holderfield, "Robust Software," *IEEE Internet Computing*, vol. 6, no. 2, Mar./Apr. 2002, pp. 78–80.
6. L.M. Stephens and M.N. Huhns, "Consensus Ontologies: Reconciling the Semantics of Web Pages and Agents," *IEEE Internet Computing*, vol. 5, no. 5, Sept./Oct. 2001, pp. 92–95.

Michael N. Huhns is a professor of computer science and engineering at the University of South Carolina, where he also directs the Center for Information Technology.